How To Use This Book

The following topics explain how to use this book:

- Using the Contents
- Getting Additional Information
- Using Menu Bar Choices
- Documentation Conventions

Using the Contents

When the Contents window is first displayed, some topics have a plus (+) sign beside them. The plus sign indicates that additional topics are available.

To expand the Contents if you are using a mouse, click on the plus sign. If you are using the keyboard, use the Up or Down Arrow key to highlight the topic, and press the Plus (+) key. For example, **Toolkit Roadmap** has a plus sign beside it. To see additional topics for that heading, click on the plus sign or highlight that topic and press the Plus (+) key.

To view a topic, double-click on the topic (or press the Up or Down Arrow key to highlight the topic, and then press Enter).

Getting Additional Information

After you select a topic, the information for that topic is displayed in a window. Highlighted words or phrases indicate that additional information is available. You will notice that certain words and phrases are highlighted in blue letters, or in white letters on a black background. These are called *hypertext* terms. If you are using a mouse, double-click on the highlighted word. If you are using a keyboard, press the Tab key to move to the highlighted word, and then press Enter. Additional information is then displayed in a window.

Using Menu Bar Choices

Several choices are available for managing information presented in this book. There are three menus on the menu bar:

- Services menu
- Options menu
- Help menu

Services Menu

The actions that are selectable from the Services menu operate on the active window currently displayed on the screen. These actions include the following:

- **Search**
- **Print**
- **Bookmark**
- **Copy**

Search

Search enables you to find occurrences of a word or phrase in the current topic, selected topics, or all topics. You can specify a word or phrase to be searched. You can also limit the search to a set of topics by first marking the topics in the Contents list.

To search for a word or phrase in all topics, do the following:

1. Select **Search** from the Services menu. Type the word or words to be searched for.
2. Click on All sections or press the Up or Down Arrow keys to select it.
3. Click on **Search** (or select it and press Enter) to begin the search.
4. The list of topics where the word or phrase appears is displayed.

Print

Print enables you to print one or more topics. You can also print a set of topics by first marking the topics in the Contents list. To print the document contents list do the following:

1. Select **Print** from the Services menu.
2. Click on **Contents** (or press the Up or Down Arrow key to select it).
3. Click on **Print** (or select it and press Enter).
4. The Contents list is printed on your printer.

Bookmark

Bookmark enables you to set a placeholder so you can retrieve information of interest to you. When you place a bookmark on a topic, it is added to a list of bookmarks you have previously set. You can view the list, and you can remove one or all bookmarks from the list. If you have not set any bookmarks, the list is empty. To set a bookmark, do the following:

1. Select a topic from the Contents.
2. When that topic is displayed, select **Bookmark** from the Services menu.
3. If you want to change the name used for the bookmark, type the new name in the field.
4. Click on the Place radio button or press the Up or Down Arrow key to select it.
5. Click on **OK** and press Enter. The bookmark is then added to the bookmark list.

Copy

Copy enables you to copy a topic that you are viewing to the System Clipboard or to a file that you can edit. You will find this particularly useful for copying syntax definitions and program samples into the application that you are developing.

You can copy a topic that you are viewing in two ways:

- **Copy** copies the topic that you are viewing into the System Clipboard. If you are using a Presentation Manager editor (for example, the System Editor) that copies or cuts (or both) to the System Clipboard and pastes to the System Clipboard, you can easily add the copied information to your program source module.
- **Copy to file** copies the topic that you are viewing into a temporary file named TEXT.TMP. You can later edit that file by using any editor. You will find TEXT.TMP in the directory where your viewable document resides.

To copy a topic, do the following:

1. Expand the Contents list and select a topic.
2. When the topic is displayed, select **Copy to file** from the Services menu.
3. The system puts the text pertaining to that topic into the temporary file named TEXT.TMP.

For information on one of the other choices in the Services menu, highlight the choice and press the F1 key.

Options Menu

The actions that are selectable from the Options menu allow you to change the way your Contents list is displayed. To expand the Contents and show all levels for all topics, choose **Expand all** from the Options menu. You can also press Ctrl+*. For information on one of the other choices in the Options menu, highlight the choice and press the F1 key.

Help Menu

The actions that are selectable from the Help menu allow you to select different types of help information. You can also press the F1 key for help information about the Information Presentation Facility (IPF).

Documentation Conventions

Throughout *Using Your Toolkit*, the following conventions distinguish the different elements of text:

plain text	Function names, structure names, data type names, message names, enumerated types, and constant names.
Initial capitalization	Key names, group-box controls, drop-down list boxes, dialog windows, combo-boxes, single-line entry (SLE) and multiple-line entry (MLE) fields.
UPPERCASE	File names and error codes.
`monospace`	Programming examples and user input at the command line prompt or into an entry field.
bold	Push buttons, check boxes, radio buttons, spin buttons, menu bar choices, and menu items.
italics	Parameters, structure fields, titles of documents, and first occurrences of words with special meaning.

Introduction

Welcome to the IBM Developer's Toolkit for OS/2 Warp (OS/2 Warp Toolkit). This book documents the following:

- Additions and updates made to the OS/2 Warp Toolkit as listed in the What's New section
- Descriptions of online documentation, code samples, and tools
- Programming considerations that have been changed or added
- Toolkit roadmap that covers the folder hierarchy and the contents and directory structure
- Toolkit support information that assists you in installing or using the OS/2 Warp Toolkit, or reporting suspected system defects as a result of installing the OS/2 Warp Toolkit

---

README

Toolkit Desktop Objects

To restore your Toolkit Desktop objects, insert the CD or diskette that contains the Toolkit Installation program (TKINSTAL.EXE), start the program, and follow these steps:

1. Click on **Options** to open **Installation options**.
2. Deselect **Install selected files** and **Write CONFIG.SYS updates to:** (only **Register WPS classes** and **Create desktop objects** should be selected).
3. Click on **OK** to close **Installation options**.
4. Highlight the root (top-most) component in the component tree, and then fill in the **Destination:** field with the location of your Toolkit subdirectory.
5. Click on **Install**.

What's New

What's New provides information about the latest release of the OS/2 Warp Toolkit.

In Version 4.5, Revision 2

The following sections describe the updates that were made in Revision 2 of the Toolkit.

Development Tools

Base Tools

Executable File Header Utility (EXEHDR)

- Two new options were added to EXEHDR: **/STACKDOS** and **/STACKOS2**. See the *Tools Reference* book for more information on these new options.

View and Set Executable Attributes (MARKEXE)

- A description of the **SETVERSION** option was added to the MARKEXE reference section in the *Tools Reference* book.

Program Maintenance Utility (NMAKE)

- A fix was made so that recursive invocations of NMAKE (normally made via the $(MAKE) macro) will always be able to locate the executable module from which NMAKE was loaded, even if the PATH environment variable was altered by the makefile and no longer references the location where the module was originally found.
- NMAKE now has the ability to set the extended LIBPATH values (BEGINLIBPATH and ENDLIBPATH) in the same way that the system command processor (CMD.EXE) does. For more information on BEGINLIBPATH and ENDLIBPATH, see the description of the **DosQueryExtLIBPATH()** and **DosSetExtLIBPATH()** API's in the *Control Program Programming Guide and Reference* .

Toolkit Information

The following documents were added in this release of the Toolkit:

Linear Executable Module Format Reference

Object Module Format Reference

In Version 4.5, Revision 1

The following sections describe the updates that were made in Revision 1 of the Toolkit.

Installation Program

On the **Installation options** dialog (accessed by selecting the **Options** button from the main panel) the "Register WPS classes" option is no longer selected by default.

Development Tools

Base Tools

Macro Assembler (ALP)

- If a symbol was declared with a PUBLIC directive then subsequently used in the name field of a segment directive, a trap condition could occur. This has been corrected.
- The **-Fd**, **-Fdd**, and **-Fed** options have been added to control the creation of "Make" dependency files.
- Support for the Pentium III instruction set was added.
- When the MEDIUM memory model is used, the predefined **@DataSize** symbol was returning 0 instead of 1. This behavior has been corrected.

Executable File Header Utility (EXEHDR)

- A fix was made to prevent the following error from occuring during the processing of certain very large modules:

  EXEHDR: error U1107: unexpected end of resident/nonresident name table.

Information Presentation Facility Compiler (IPFC)

- The compiler no longer prints consecutive dot characters (.) to the standard output for each cell item that is processed.

Legacy Resource Compiler (RC16)

- The compiler no longer prints consecutive dot characters (.) to the standard output for each resource that is processed.

Message Segment Binder (MSGBIND)

- MSGBIND no longer prints the number of each message to the standard output during the bind operation.
- Certain executable modules created by the version of ILINK shipped with the *IBM C/C++ Compilers Version 3.6* were being corrupted by MSGBIND. This has been corrected.

Resource Compiler (RC)

- The compiler no longer prints consecutive dot characters (.) to the standard output for each resource that is processed.
- The Resource Compiler was updated so that it does not alter the character case of the filenames it is processing. Previous versions converted filenames to consist of all uppercase characters.

Toolkit Information

ALP Programming Guide and Reference

Documentation has been added to describe new command line options and support for the Pentium III instruction set.

In Version 4.5

- The following sections provide information about updates and additions:
  - TCP/IP Toolkit Enhancements
  - New APIs in OS/2 Warp Server for e-business
  - Debugging and Serviceability Enhancements

- The following tools have been removed from the Toolkit:
  - IBM no longer supports SOM; however, the information about SOM remains in the Toolkit.
  - The ICAT Debugger is no longer included in the Toolkit; however, you can download the debugger from the following Web site:

    http://service.boulder.ibm.com/icat/
  - OpenDoc is no longer included in the Toolkit; however, information concerning OpenDoc is available at the following Web site:

    http://www.software.ibm.com/ad/opendoc/
  - The Universal Resource Editor (URE) is no longer included in the Toolkit. This tool has been replaced by the Integrated Resource Editor (IRE), which is included in IBM C++ Professional Version 4.0. Information about this product is available at the following Web site:

    http://www.ibmlink.ibm.com

You should also review Important Information before using the tools and sample programs in the Toolkit.

TCP/IP Toolkit Enhancements

The TCP/IP Toolkit has been upgraded to version 4.21. Enhancements to the TCP/IP Toolkit are:

- New APIs:
  - accept_and_recv()
  - send_file
- A new 32-bit version of the ring-0 library (R0LIB32), which enables device drivers to call the new 32-bit networking kernel (socket.sys) without the 16-bit to 32-bit thunk layer.

For information about the new TCP/IP samples added to this Toolkit, refer to TCP/IP Samples .

New APIs in OS/2 Warp Server for e-business

OS/2 Warp Server for e-business includes some new and some revised APIs. These APIs relate to control programming functions, particularly those that are affected by Logical Volume Manager (LVM), and to performance and RAS functions. For more information, refer to the online book, *OS/2 Programming Guide and Reference Addendum for OS/2 Warp Server for e-business.*

Debugging and Serviceability

Major enhancements to the system diagnostic tools (RAS Enhancements) are provided in FixPak 35 for OS/2 Warp 3.0, FixPak 10 for OS/2 Warp 4.0, and OS/2 Warp Server for e-business. They provide major functional enhancements to the System Trace and Process Dump facilities, a summary of which is given below. Full details of these enhancements are provided in the following documents that are included with the operating system:

- OS2\SYSTEM\RAS\TRACE.DOC
- OS2\SYSTEM\RAS\DTRACE.DOC
- OS2\SYSTEM\RAS\PROCDUMP.DOC
- OS2\SYSTEM\RAS\TRSPOOL.DOC

Detailed information on debugging techniques and the use of system debugging is provided in *The OS2 Debugging Handbook (SG244640.INF),* which is distributed with this TOOLKIT.

Any changes and enhancements to serviceability tools introduced with a FixPak are documented \OS2\INSTALL\README.DBG.

Important Information

This section describes some items that can affect your development efforts when using the OS/2 Toolkit.

Compiling with IBM C/2

The OS/2 Toolkit for OS/2 Warp Version 4.0 and later versions of the Toolkit are not intended to be used with the IBM C/2, Version 1.1 compiler. If you choose to use that compiler, you should use version 1.3 of the Toolkit.

The OS/2 Warp Toolkit is intended to be used with the IBM VisualAge C++ compiler.

Compiling with IBM VisualAge C++ for OS/2

The IBM VisualAge C++ compiler version 3.0 includes its own version of the OS/2 Warp Toolkit, which includes updated sample makefiles. Changes to the sample makefiles relate to the new linker (ILINK) and to the new library (CPPOM30.LIB).

With this version of the OS/2 Warp Toolkit, the same sample makefile changes have been made so that you can use VisualAge C++.

You can compile the samples included with the OS/2 Warp Toolkit with C Set ++ (using LINK386) by changing any reference to CPPOM30.LIB to:

```
DDE4MBS.LIB
```

and changing any reference to ILINK to:

```
LINK386
```

Note: The */nofree* option must be removed from the LINK386 statement.

Installing IBM VisualAge C++ for OS/2

Installing your compiler first and the OS/2 Warp Toolkit last prevents unexpected results due to environment variable changes (automatic updates to CONFIG.SYS).

To avoid a downlevel SOM Runtime installed by the VisualAge C++ 3.0 compiler, follow these steps for installation:

1. Start the installation program for VisualAge C++ 3.0.
2. Deselect the OS/2 Warp Toolkit entries in the installation screen and proceed with the installation.
3. After completing the installation of VisualAge C++ but before restarting the system, edit CONFIG.SYS and modify the LIBPATH statement by moving x:\ IBMCPP\DLL after y:\OS2\DLL, where *x* is the drive where the VisualAge C++ compiler is installed and *y* is the drive where the system files for OS/2 Warp Version 4 or later are installed. This prevents the SOM Runtime installed by VisualAge C++ from overwriting the SOM Runtime shipped with OS/2 Warp.
4. Save the modified CONFIG.SYS and restart your system.
5. Install the the OS/2 Toolkit and restart your system.

VoiceType Developer's Toolkit Updates

The header files and libraries, samples, tools, and documentation associated with the VoiceType Developer's Toolkit are located in the \TOOLKIT\SPEECH subdirectories.

The VoiceType Developer's Toolkit includes:

- Grammar compiler and its associated documentation
- Utilities to be used with grammars
- SMAPI.DLL

Toolkit Contents

This section describes the contents of the OS/2 Warp Toolkit. It contains the following topics:

- Toolkit Roadmap -- provides the directory hierarchy and the folder hierarchy of the Toolkit.
- Header Files -- describes special programming considerations for various header files included with the OS/2 Warp Toolkit.
- Sample Programs -- provides descriptions of the code samples that are included in the OS/2 Warp Toolkit.
- Tools -- provides descriptions of the Presentation Manager, Multimedia, SOM, TCP/IP, Workplace Shell, and XPG4 tools. It also introduces you to the interface that installs the debug kernel, symbol files, and debug version of PM and Workplace Shell and describes tools that support your debugging efforts.
- Online Documentation -- describes the online books of the OS/2 Warp Toolkit.

Toolkit Roadmap

The tools, samples, and technical documentation in the OS/2 Warp Toolkit are available in two ways. You can access them through the Toolkit folder and its subfolders on the Desktop, or you can access them using an OS/2 window or OS/2 full-screen session and changing into the \TOOLKIT subdirectory and its subdirectories.

Directory Hierarchy

The OS/2 Warp Toolkit package has the following directory structure:

```
\TOOLKIT     - Root subdirectory for the Toolkit

    \BIN     - Programming tools
    \BITMAP  - Sample multimedia bitmaps
    \BOOK    - Online technical information
    \DLL     - Toolkit dynamic link library (DLL) files
    \H       - C, C++, and DTS header files
    \HELP    - Toolkit help (HLP) files
    \ICON    - Toolkit icon (ICO) files
    \IDL     - Workplace Shell interface definition language (IDL) files
    \INC     - Assembler header (INC) files
    \IPFC    - Information Presentation Facility Compiler (IPFC) files
    \LIB     - Import library (LIB) files
    \SAMPLES - Samples  programs
    \SOM     - SOM subdirectories
    \SPEECH  - VoiceType Developer's Toolkit
```

Folder Hierarchy

The OS/2 Warp Toolkit package has the following Workplace Shell folder hierarchy:

```
IBM Developer's Toolkit            - Root folder of the Toolkit
for OS/2 Warp                        (contains the README file and the
                                     other Toolkit folders)

   Development Tools               - Programming tools
   Multimedia Bitmaps              - Sample multimedia bitmaps
   Multimedia Sample Programs      - Multimedia sample programs
   Open32 Samples                  - Open32 sample programs
   OpenGL Samples                  - OpenGL sample programs
   OS/2 Sample Programs            - Control Program sample programs
   PM Sample Programs              - Presentation Manager (PM) sample programs
   REXX Sample Programs            - REXX sample programs
   Toolkit Information             - Online technical books
   VoiceType Developer's Toolkit   - VoiceType tools, samples, and online books
   Workplace Shell Sample Programs - Workplace Shell (WPS) sample programs
```

Header Files

You should be aware of special programming considerations in the following sets of header files:

- Control Program
- Multimedia
- OS/2

Note: For compatibility with VisualAge C++, the *#pragma checkout* directives in all header files have been changed to *#pragma info* . The *#pragma checkout* directive is no longer supported by VisualAge C++. Because the *#pragma info* directive provides similar function to *#pragma checkout* , the header file changes should not affect code that uses the header files. If you use the *#pragma checkout* directive with VisualAge C++, it generates an "unsupported pragma" error for both C and C++ compilers. Refer to the *VisualAge C++ Language Reference* for more details.

Control Program Header Files

DosSetDOSProperty() and DosQueryDOSProperty() in BSEDOS.H are not supported. Do not use these functions.

Multimedia Header Files

The multimedia header files listed below are delivered in two different versions. One version uses conventions compatible with the standard OS/2 header-file format. The other version uses conventions compatible with Microsoft Windows header files. The Windows-style headers are currently included for compatibility with earlier multimedia applications, but will be removed from the Toolkit in the future.

Use the OS/2-style header files for new applications and modify the source code for existing applications that use the Windows-style headers.

Windows-Style	**OS/2-Style**
CDAUDIO.H	CDAUDOS2.H
MCIDRV.H	MMDRVOS2.H
MIDI.H	MIDIOS2.H
MMIO.H	MMIOOS2.H
MMSYSTEM.H	MCIOS2.H

OS/2 Header Files

Instead of using the SELECTOROF macro in the OS2DEF.H header file, use one of the following two macros to obtain the selector of a 16:16 address :

```
#define SELECTOROF(p)   (((PUSHORT)&(p))[1])
OR
#define SELECTOROF(p)   ((ULONG)(p)>>16)
```

If you use the selector returned from one of the above macros with the OFFSETOF and MAKEP macros in the OS2DEF.H header file, you can successfully convert a 16:16 address to a 0:32 address.

Sample Programs

This section describes the sample programs available with the OS/2 Warp Toolkit.

The sample programs are categorized as follows:

- BIDI (Bidirectional)
- LAN Server
- Multimedia
- Open32
- OpenGL
- OS/2
- Presentation Manager
- Problem Determination
- REXX
- TCP/IP
- Workplace Shell

Most sample programs are written in C language and demonstrate the use of the functions of the control program (CP, base OS/2 operating

system), the PM interface, the multimedia (MM) interface, and the Workplace Shell (WPS).

Each sample program serves as a template that can be easily modified for your own purposes. These programs let you investigate the best way to implement your own application requirements.

Some of the sample programs require specific hardware devices. When appropriate, the hardware is listed following the program descriptions. Without these devices, you can still compile and run the sample programs; however, you might not receive the full effect of the program. For example, if a multimedia sample program has audio, you will not hear it unless you have an audio adapter supported by OS/2 multimedia and speakers installed.

Most samples also contain the overhead routines necessary to create a PM application, as well as stubs for the basic menu items that all applications should have.

There are many comments within the source code that clarify technical information.

Note: Names of the sample programs, in most cases, correspond to their Toolkit subdirectory names.

Starting Sample Programs

There are two ways to start a sample program:

- From the Desktop:

  When installed, all sample programs (with the exception of the bidirectional samples, TCP/IP samples, Problem Determination samples, LAN Server samples, some of the Multimedia samples, and some of the Workplace Shell samples) appear in the sample programs folders located within the Toolkit folder. To start a sample program, open the Toolkit folder, open the folder representing the type of samples you wish to execute, and then select the appropriate sample program.

- From an OS/2 command prompt:

  Change to the subdirectory where the sample is located, type the name of the executable file and press Enter.

Most samples include a description file (or makefile) that you can use to build the sample yourself. The name of the makefile is either MAKEFILE or *filename* .MAK, where *filename* refers to the name of the sample. To build the sample,

1. Go to an OS/2 command prompt and change to the subdirectory where the makefile is located.

   You must execute these commands from the subdirectory where the makefile is located in order to build the sample correctly.

2. Enter one of the following, depending on the name of the makefile:

   - When the name of the makefile is MAKEFILE, type:

     ```
     NMAKE
     ```

   - When the name of the makefile is *filename* .MAK, type:

     ```
     NMAKE /f filename.MAK
     ```

For information about the NMAKE tool, refer to the *Tools Reference* .

Note: The makefile assumes that you are using the IBM VisualAge C++ for OS/2 compiler for code compilation.

BIDI Sample Programs

These samples demonstrate the use of the Bidirectional Language support available in the Arabic and Hebrew versions of OS/2. There are

two sets of samples provided: one set for the Arabic language and the other set for the Hebrew language. For both languages, there are two samples:

- STYLE
- TELDIR

Note: Programs that use the bidirectional support functions will run only on the Arabic and Hebrew versions of the OS/2 operating system. These are currently the only versions of the operating system that support the bidirectional support functions.

STYLE Sample Program

STYLE is a sample program that demonstrates the bidirectional language support features of PM controls and other system components.

Note: The Arabic versions of this sample (located in \TOOLKIT\SAMPLES\BIDI\ARABIC), require the Arabic version of OS/2 to run, and the Hebrew versions of this sample (located in \TOOLKIT\SAMPLES\BIDI\HEBREW), require the Hebrew version of OS/2 to run.

TELDIR Sample Program

TELDIR is a simple bilingual telephone directory application that demonstrates how language and orientation selections can be set dynamically by an application that uses bidirectional functions.

Note: The Arabic versions of this sample require the Arabic version of OS/2 to run, and the Hebrew versions of this sample require the Hebrew version of OS/2 to run.

Device Driver Sample Programs

Physical device driver (PDD) and virtual device driver (VDD) samples are not included in the OS/2 Warp Toolkit. See the Developer Connection Device Driver Kit for OS/2 for device driver samples.

LAN Server Sample Programs

The LAN Server sample programs are as follows:

- ACCESS32 - Access Control Profile Sample
- ALIAS32 - File Resource Alias Sample
- FILEEN32 - Server Open Files Sample
- MESSAG32 - Network Message Sample
- MLOGON32 - Network Logon Sample
- SADMIN32 - Server Administration Sample
- USER32 - User Account Sample

ACCESS32

ACCESS32 (located in \TOOLKIT\SAMPLES\LANSERV\ACCESS32) returns all the access control profiles (ACPs) for a particular resource. You may alter any given ACP by specifying the user and the new permissions.

ACCESS32 demonstrates how to use the following NetAPI functions: Net32AccessAdd, Net32AccessGetInfo, and Net32AccessSetInfo.

This program can operate both locally and remotely. The machine running this program must be logged on to the LAN with administrator authority.

Starting ACCESS32

To query a resource's ACP, type:

```
access32 <resource>
```

To change the ACP for a user, type:

```
access32 <resource> <name:permissions> <name:permissions> ...
```

ALIAS32

ALIAS32 (located in \TOOLKIT\SAMPLES\LANSERV\ALIAS32) adds a file resource alias and shares it, gives the USERS group read-only access to the resource, and adds a directory limit to the resource.

ALIAS32 demonstrates how to use the following NetAPI functions:

Net32WkstaGetInfo
Net32ServerEnum2
Net32GetDCName
Net32AliasAdd
Net32AliasDel
Net32ShareAdd
Net32AccessAdd
Net32AccessGetInfo
Net32AccessSetInfo
Net32DASDAdd

Starting ALIAS32

To start ALIAS32 from a command line, type:

```
alias32 aliasname resourcename
```

or

```
alias32 aliasname servername resourcename
```

where:

aliasname Is an 8-byte name that is not currently in use.

servername Specifies the server that the alias resource resides on. If not specified, it is assumed that the alias resource resides on the local machine.

resourcename Is a fully qualified path name.

ALIAS32 checks that the device is shared and returns basic information on the device type.

FILEEN32

FILEEN32 (located in \TOOLKIT\SAMPLES\LANSERV\FILEEN32) returns information on open files on a specific server. You can choose to view a list of all open files or to restrict the list by device path or user. You can leave both the path and the user ID unspecified to get information on all files opened by all users on the specified server.

FILEEN32 demonstrates how to use the following NetAPI function: Net32FileEnum2.

Starting FILEEN32

To start FILEEN32 for a remote server, type:

`fileen32 servername basepath`

To start FILEEN32 for a local server, type:

`fileen32 basepath username`

where:

servername Indicates the machine where open files will be enumerated. If a *servername* is specified, it must be the first parameter specified. If no *servername* is specified, open files at the local server will be enumerated.

basepath Is in the form of:

`drive:\pathname`

Only open files matching this *basepath* will be enumerated. If *basepath* is not specified, all open files are enumerated.

username Is the name of an existing user. If *username* is specified, only files opened by this user are enumerated. The *username* must be specified as the last parameter.

If no parameters are specified, all open files at the local server are enumerated.

This program must be run by an administrator. The opened files that are to be listed must be on a redirected drive on the target server.

MESSAG32

MESSAG32 (located in \TOOLKIT\SAMPLES\LANSERV\MESSAG32) is designed to send simple messages through the messenger service. You can send messages *from* the local machine or a remote server. You can send messages *to* specific users or a domain group, or broadcast to an entire LAN.

MESSAG32 uses Net32MessageBufferSend to send the message.

Starting MESSAG32

To start MESSAG32 from the command line, type:

```
messag32
```

MESSAG32 prompts the user for where the message should be sent from:

```
1 - the local machine
2 - a remote server
```

If the message is being sent from a remote server, MESSAG32 prompts the user for a servername. MESSAG32 then prompts the user for who the message should be sent to as follows:

```
1 - a user
2 - a domain
3 - all the requesters on the LAN
```

If the message is being sent to a user or domain, MESSAG32 prompts the user for the name of the user or domain to send the message to. Finally, MESSAG32 prompts the user to enter a text string for the message.

MLOGON32

MLOGON32 (located in \TOOLKIT\SAMPLES\LANSERV\MLOGON32) logs on a user, waits for the user to press the Spacebar, and logs off the user. The user name, password (if required), and domain name are passed as command line parameters to the program.

Starting MLOGON32

To start MLOGON32 from the command line, type:

```
mlogon32 username password domainname
```

or

```
mlogon32 username domainname
```

SADMIN32

SADMIN32 (located in \TOOLKIT\SAMPLES\LANSERV\SADMIN32) executes a command on a remote or local server. The command to be executed on the server is passed as a command line parameter to this program, but it must reside on a disk visible to the target server.

SADMIN32 demonstrates how to use NetServerAdminCommand, which is a NetAPI function.

Starting SADMIN32

To start SADMIN32 from the command line, type:

```
sadmin32 \\targetservername command
```

USER32

USER32 (located in \TOOLKIT\SAMPLES\LANSERV\USER32) is designed to access and control user account information. USER32 can display information about a user, delete a user, or create a new user. When a new user is created, USER32 creates a new alias, gives the user access to it, and makes it a logon assignment for the user; creates a new group and adds the user to it; creates a private application definition and assigns that too to the user.

USER32 demonstrates how to use the following NetAPI functions:

Net32AccessAdd
Net32AccessGetInfo
Net32AccessSetInfo
Net32AliasAdd
Net32AliasDel
Net32AliasGetInfo
Net32AppAdd
Net32GetDCName
Net32GroupAdd
Net32GroupAddUser
Net32GroupDel
Net32GroupGetInfo
Net32ServerEnum2
Net32WkstaGetInfo
Net32UserAdd
Net32UserDCDBInit
Net32UserDel
Net32UserGetGroups
Net32UserGetInfo
Net32UserGetLogonAsn
Net32UserSetAppSel
Net32UserSetLogonAsn

NetStatisticsGet2 retrieves information from both the requester and the server portions of a server machine. Note that these two sets of information, even when they are in the same machine, are quite different.

Starting USER32

To display information about a user, type:

```
user32 username
```

To delete a user, type:

```
user32 username /DELETE
```

To create a new user, type:

```
user32 username /ADD
```

where *username* is the user ID of the account.

Multimedia Sample Programs

The Multimedia sample programs are as follows:

- ADMCT
- ASYMREC
- AVCINST
- BEEHIVE
- CAPSAMP
- CAPTION
- CASECONV
- CDMCIDRV
- Control File Templates
- CLOCK
- CODEC
- DAUDIO
- DIVE
- DOUBPLAY
- DUET1
- DUET2
- FSDIVE
- FSSHT
- MCD Command Tables
- MCDTEMP
- MCISPY
- MCISTRNG
- MIDISAMP
- MMBROWSE
- MMOTTK
- MOVIE
- RECORDER
- Short Control File Templates
- SHRC
- TUNER
- ULTIEYES
- ULIOT

ADMCT Sample Program

ADMCT (located in \TOOLKIT\SAMPLES\MM\ADMCT) is an example of a media control driver (MCD) that demonstrates how to control a streaming device. Streaming devices use the services of the sync/stream manager (SSM) of OS/2 multimedia to control the data stream from a source location to a target location.

ASYMREC Sample Program

ASYMREC (located in \TOOLKIT\SAMPLES\MM\ASYMREC) illustrates how to include asymmetric recording function in your multimedia application. Modules include source code extracted from the Video IN recorder application, which enables frame-step recording using Ultimotion compression techniques.

AVCINST I/O Procedure Sample

AVCINST (located in \TOOLKIT\SAMPLES\MM\AVCINST) illustrates how an application can install and remove an I/O procedure to use multimedia input/output (MMIO) file services. The AVC I/O procedure installation sample is a simple PM application that enables you to install or remove the audio AVC I/O procedure, AVCAPROC.DLL.

Hardware requirements:

- Computer capable of running OS/2 Warp

Software requirements:

- OS/2 Warp
- Multimedia support

BEEHIVE Sample Program

BEEHIVE demonstrates the power of compiled sprites by comparing the performance of a typical sprite algorithm against a compiled sprite.

When the application first starts, it will display a single sprite moving in a random path around the application window. You can add sprites by holding down the H key. As the sprite count increases, the effects on the frame rate will become evident. The actual frame rate will be displayed on the title bar.

You will be able to switch the application from compiled sprites to normal sprites using a menu. As the number of sprites increases, the difference in performance from compiled to normal sprites becomes more dramatic.

While DIVE offers a high-performance API for blitting images to the screen, it was not designed for implementing sprites. BEEHIVE demonstrates a technique for implementing high-performance sprites in an OS/2 Warp/DIVE environment. The source code provided offers a good starting point for building a high-performance sprite engine.

Hardware Requirements:

- Computer capable of running OS/2 Warp
- SVGA (a screen mode of 640x480x256 constitutes a minimum SVGA system)

Software Requirements:

- OS/2 Warp Version 3 or later
- Multimedia support

Starting BEEHIVE

You can start BEEHIVE in two ways:

- From the command line, change to the TOOLKIT\SAMPLES\MM\BEEHIVE subdirectory and type:

  ```
  BEEHIVE
  ```

- From the Desktop, open the Toolkit folder and then:

  1. Open the Multimedia Samples folder.
  2. Open the Video Samples folder.
  3. Double-click on **Beehive**.

CAPSAMP Sample Program

CAPSAMP (located in \TOOLKIT\SAMPLES\MM\CAPSAMP) and the caption utility functions (located in \TOOLKIT\SAMPLES\MM\CAPDLL) are part of the sample captioning system provided with the OS/2 Warp Toolkit. CAPSAMP demonstrates how captioning can be integrated into applications using caption files in conjunction with the caption utility functions.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Speaker or headphones
- Sound card

Software requirements:

- OS/2 Warp
- Multimedia support

CAPTION Sample Program

CAPTION (located in \TOOLKIT\SAMPLES\MM\CAPTION) is part of the sample captioning system provided with the OS/2 Warp Toolkit. The caption creation utility program creates a "caption" file. It enables you to synchronize an audio file with a text file.

This concept can be extended beyond audio and text to apply to many possibilities, such as synchronizing audio and video or synchronizing video and text.

Note: In order for the Caption sample to work correctly, captioning must be enabled in the operating system. To enable captioning proceed as follows:

- Open the Multimedia Setup object in the Multimedia folder
- Go to the System page
- Place a check mark in the Captioning check box.

CASECONV I/O Procedure Sample

CASECONV (located in \TOOLKIT\SAMPLES\MM\CASECONV) provides a simple example of how to write a file format I/O procedure (without illustrating the use of data translation). This sample performs case conversion of text.

CDMCT Sample Program

CDMCT (located in \TOOLKIT\SAMPLES\MM\CDMCIDRV) is an example of a media control driver (MCD) that demonstrates how to control a non-streaming device. Non-streaming devices stream data within the device.

Control File Templates

The \TOOLKIT\SAMPLES\MM\CF subdirectory contains control file templates you can utilize when installing a program using MINSTALL.

CLOCK Sample Program

CLOCK (located in \TOOLKIT\SAMPLES\MM\CLOCK) illustrates the use of the memory playlist feature of OS/2 multimedia. The memory playlist feature provides for easy manipulation of multimedia in memory to create unique effects based on user input or other dynamic events.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Speaker or headphones
- Sound card

Software requirements:

- OS/2 Warp
- Multimedia support

CODEC Sample Program

CODEC (located in \TOOLKIT\SAMPLES\MM\CODEC) illustrates how to write a CODEC procedure to include compression and decompression routines in your multimedia applications. A CODEC procedure operates on data within a file or buffer.

DAUDIO Sample Program

DAUDIO (direct audio) demonstrates the use of the direct audio interface. This high speed audio interface enables an application to send audio data directly to the amp-mixer device. The sample demonstrates the steps required to set up and use this new interface for playing and recording digital audio data.

See the *Multimedia Application Programming Guide* and the *Multimedia Programming Reference* for more information on the direct audio interface.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Sound card
- Speakers or headphones

Software requirements:

- OS/2 Warp Version 4 or later
- Multimedia support

Starting DAUDIO

You can start DAUDIO in two ways:

- From the command line, change to the TOOLKIT\SAMPLES\MM\DAUDIO subdirectory and type:

  ```
  DAUDIO
  ```

- From the Desktop, open the Toolkit folder and then:

  1. Open the Multimedia Samples folder.
  2. Open the Audio Samples folder.
  3. Double-click on **Direct Audio**.

DIVE Sample Program

DIVE (located in \TOOLKIT\SAMPLES\MM\DIVE) illustrates the use of the direct interface video extensions. DIVE provides optimized blitting performance for motion video subsystems and applications that perform rapid screen updates in the OS/2 PM and full-screen environments. Using DIVE interfaces, applications can either write directly to video memory or use the DIVE blitter. The DIVE blitter takes advantage of acceleration hardware when present and applicable to the function being performed.

Note: The DIVE sample requires OS/2 Warp Version 3 or later in order to execute properly. The files for the samples will be installed when the samples are selected, but Workplace Shell objects will not be created for them if the installed operating system is not OS/2 Warp Version 3 or higher.

The OS/2 Warp color support defaults to 16 colors. This means that your setup needs to be updated; otherwise, the DIVE sample will not run.

The maximum window size of this sample has been limited to 640x480 because larger window sizes can cause excessive swapping on machines with less than 16MB.

Hardware requirements:

- Computer capable of running OS/2 Warp
- SVGA (a screen mode of 640x480x256 constitutes a minimum SVGA system)

Software requirements:

- OS/2 Warp
- Multimedia support
- Software motion video

DOUBPLAY Sample Program

DOUBPLAY (located in \TOOLKIT\SAMPLES\MM\DOUBPLAY) enables an application to play audio files directly from application memory buffers. An array of multiple playlist structures that combines playlist commands with data buffers can be constructed to perform complex operations on multiple buffers.

This sample takes a single wave file, MYWAVE.WAV, and plays it using a memory playlist. The playlist is constructed as a circular buffer

composed of a series of small buffers, the sum of which might be larger than the size of the .WAV file. This circular buffer is used to repeatedly play an audio file when the PLAY button is selected. As each buffer in the playlist is expended, the application refills the expended buffer with new data from the .WAV file. When all the buffers in the playlist have been expended, the playlist branches back to the first buffer to play the new data. This circular buffering process continues until the STOP button is selected or the application is closed.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Speakers or headphones
- Sound card

Software requirements:

- OS/2 Warp
- Multimedia support

DUET1 Sample Program

DUET1 (located in \TOOLKIT\SAMPLES\MM\DUET1) illustrates the OS/2 multimedia concept of device grouping and integrating multimedia into an application's help information. This sample demonstrates the concepts of grouping two streaming devices.

Note: The Streaming Device Duet sample (DUET1, located in \TOOLKIT\SAMPLES\MM\DUET1) requires that the IBM M-Audio Capture and Playback Adapter card be installed in order for the sample to execute properly.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Speaker or headphones
- Sound card (capable of playing two mono wave files simultaneously such as the IBM M-AUDIO card)

Software requirements:

- OS/2 Warp
- Multimedia support

DUET2 Sample Program

DUET2 (located in \TOOLKIT\SAMPLES\MM\DUET2) illustrates the OS/2 multimedia concept of device grouping and integrating multimedia into an application's help information. This sample demonstrates how one of the devices in the multimedia device group can be a non-streaming device.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Speaker or headphones
- Sound card
- CD-ROM drive

Software requirements:

- OS/2 Warp
- Multimedia support

FSDIVE Sample Program

FSDIVE (full-screen DIVE) demonstrates the use of multimedia's direct interface video extensions (DIVE) by repeatedly displaying a short animation sequence. The animation is performed by sequentially displaying a series of up to 16 bitmaps in a PM window. You can display the default bitmaps that are included with the sample or specify the bitmaps by passing the file names as command line parameters.

After the application is started, you can move or resize the window and observe the effects on the frame rate of the animation (displayed on the title bar).

The latest version of the DIVE interface has been enhanced to allow an application to take over the display and change the resolution. This enables an application to run in a full screen without paying the performance penalty of maintaining a high-resolution Desktop. Full-screen DIVE can be activated by using the Alt+Home hot key.

Note: To run the full-screen DIVE sample on an OS/2 Warp Version 3 system, you must first install full-screen support by running GSRVINST.EXE (located in \TOOLKIT\SAMPLES\MM\FSDIVE). See the GAMESRVR.DOC file in the same directory for more details. This installation is not necessary on an OS/2 Warp Version 4 or later system.

Hardware Requirements:

- Computer capable of running OS/2 Warp
- SVGA (a screen mode of 640x480x256 constitutes a minimum SVGA system)

Software Requirements:

- OS/2 Warp Version 3 or later
- Multimedia support
- Software motion video

The maximum window size of this sample has been limited to 640x480 because larger window sizes can cause excessive swapping on machines with less than 16MB.

Starting FSDIVE

You can start FSDIVE in two ways:

- From the command line, change to the TOOLKIT\SAMPLES\MM\FSDIVE subdirectory and type:

  ```
  FSDIVE
  ```

- From the Desktop, open the Toolkit folder and then:

  1. Open the Multimedia Samples folder.
  2. Open the Video Samples folder.
  3. Double-click on **FS Dive**.

FSSHT Sample Program

FSSHT (located in \TOOLKIT\SAMPLES\MM\FSSHT) contains a sample file system stream handler.

MCD Command Tables

Before a media control driver (MCD) can interpret a string command, the media device manager (MDM) must use a command table to

change the string into an equivalent procedural command. Represented as resources to the driver, command tables are created using the RCDATA type of resource. The resource number of the RCDATA block is the device type number. The \TOOLKIT\SAMPLES\MM\MCDTBL subdirectory contains command tables for each of the following devices:

- Amp Mixer
- CD-ROM/XA
- CD Audio
- Digital Video
- Sequencer
- Videodisc
- Wave Audio

If you want to support device-specific messages, you must create a device- specific command table.

MCDTEMP Template

MCDTEMP (located in \TOOLKIT\SAMPLES\MM\MCDTEMP) provides a basic template to write a media control driver (MCD). Refer to the \ADMCT and \CDMCIDRV subdirectories for specific streaming or to the multimedia I/O (MMIO) samples.

MCISPY Sample Program

MCISPY (located in \TOOLKIT\SAMPLES\MM\MCISPY) monitors media control interface messages that are exchanged between applications and the OS/2 multimedia subsystem.

In addition to teaching you about multimedia messages, MCISPY also serves as a powerful debugging aid.

The MCISPY sample must be manually set up. To do so, follow these steps:

1. Copy the MDM.DLL file located in the \MMOS2\DLL subdirectory to the \TOOLKIT\DLL subdirectory.
2. Use the DLLRNAME tool (available with C Set ++ or VisualAge C++) to rename the copy to MCI.DLL. To do this, type:

   ```
   DLLRNAME MDM.DLL MDM=MCI
   ```

3. Create the executable (MCISPY.EXE) and the stub DLL (MDM.DLL) by running the MCISPY makefile.
4. Place the stub MDM.DLL, built by the MCISPY makefile, in the LIBPATH so that it is recognized prior to the MDM.DLL file located in the \MMOS2\DLL subdirectory. The source code for the stub MDM.DLL is included in the MCISPY sample.
5. Restart your system.

Note: Driver notifications might not be visible in releases prior to OS/2 Warp Version 3. These notifications include all multimedia messages routed through mdmDriverNotify().

Hardware requirements:

- Computer capable of running OS/2 Warp

Software requirements:

- Multimedia support
- OS/2 Warp

  Note: The files for the samples will be installed when the samples are selected, but Workplace Shell objects will not be created for them if the installed operating system is not OS/2 Warp.

MCISTRNG Sample Program

MCISTRNG (located in \TOOLKIT\SAMPLES\MM\MCISTRNG) serves as a powerful testing and debugging tool that enables developers, who are writing media drivers, to control their devices at the application level. The string test sample illustrates how an application uses the interpretive string interface provided by the media control interface.

This sample also illustrates how notification messages are returned from the media drivers to the application.

Hardware requirements:

- Computer capable of running OS/2 Warp

Software requirements:

- OS/2 Warp
- Multimedia support

MIDISAMP Sample Program

MIDISAMP illustrates the use of the real-time MIDI (RTMIDI) programming concepts and usage of the new RTMIDI API. This sample initializes and sets up a small MIDI node network and subsequently sends a MIDI message from an application node to a hardware node, thereby demonstrating MIDI playback.

The RTMIDI functions are documented in the *Multimedia Programming Reference* .

Hardware requirements:

- Computer capable of running OS/2 Warp
- A sound card supported by a "Type A" RTMIDI device driver

Note: Type A device drivers provide direct hardware support for RTMIDI. The following is a list of supported Type A device drivers:

- MPU-401 - A generic driver for sound cards with true hardware MPU-401 compatibility (for example, not the AWE-32). It uses only UART dumb mode. This driver is also an MMPM/2 driver. It can be found on the *DDPak* CD-ROM.

- OPL-3 - A generic driver for sound cards with true hardware Yamaha OPL-3 FM chips. This driver is also an MMPM/2 driver. It can be found on the *DDPak* CD-ROM.

- PAS-16 - A device driver for the Media Vision PAS-16 line of audio cards. This driver is also an MMPM/2 driver. It can be installed using the Selective Install program.

Software requirements:

- OS/2 Warp Version 4 or later
- Multimedia support
- Standard MIDI file

Starting MIDISAMP

You can start MIDISAMP in two ways:

- From the command line, change to the TOOLKIT\SAMPLES\MM\MIDI subdirectory and type:

```
MIDISAMP filename.mid
```

where *filename.mid* is a MIDI file.

- From the Desktop, open the Toolkit folder and then:
    1. Open the Multimedia Samples folder.
    2. Open the Audio Samples folder.
    3. Double-click on **MIDI Samp**.

Sample MIDI files are located in the MMOS2\SOUNDS subdirectory.

MMBROWSE Sample Program

MMBROWSE (located in \TOOLKIT\SAMPLES\MM\MMBROWSE) illustrates how to use the multimedia I/O (MMIO) subsystem to install I/O procedures for various image formats and then convert these image formats to OS/2 bitmaps.

Hardware requirements:

- Computer capable of running OS/2 Warp

Software requirements:

- OS/2 Warp
- Multimedia support

MMOTTK I/O Procedure Sample

MMOTTK (located in \TOOLKIT\SAMPLES\MM\MMIOPROC) provides an example of how to write an I/O procedure for use with image file formats. This sample enables file format transparency for M-Motion still video files and illustrates the use of data translation.

MOVIE Sample Program

MOVIE (located in \TOOLKIT\SAMPLES\MM\MOVIE) demonstrates device control of a software motion video device.

This sample also illustrates how to cut, copy, paste, and delete movie data from an application.

A movie can be played in an application-defined window or in the system default window provided by the software motion video subsystem.

Note: If you installed the OS/2 Warp Toolkit from diskette, the Movie sample (located in \TOOLKIT\SAMPLES\MM\MOVIE) does not contain the MOVIE.AVI file necessary to execute the application. Copy any .AVI file from the \MMOS2\MOVIES subdirectory on the drive on which you have OS/2 multimedia installed. The .AVI file should be copied to the \TOOLKIT\SAMPLES\MM\MOVIE subdirectory on which you have the OS/2 Warp Toolkit samples installed, and the target name of the file should be MOVIE.AVI.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Speakers or headphones

- Sound card
- SVGA (a screen mode of 640x480x256 constitutes a minimum SVGA system)

Software requirements:

- OS/2 Warp
- Multimedia support
- Software motion video

RECORDER Sample Program

RECORDER (located in \TOOLKIT\SAMPLES\MM\RECORDER) illustrates the concept of recording audio through the media control interface and how to query a device to find out the recording capabilities.

This sample also illustrates how to change the audio recording and audio device properties, such as bits per sample, samples per second, input level, and input source.

Hardware requirements:

- Computer capable of running OS/2 Warp
- Speaker or headphones
- Sound card

Software requirements:

- OS/2 Warp
- Multimedia support

Short Control File Templates

The \TOOLKIT\SAMPLES\MM\SHORTCF subdirectory contains a simple example of control file templates you can use when installing a program using MINSTALL.

SHRC Sample Program

SHRC (located in \TOOLKIT\SAMPLES\MM\SHRC) provides a sample stream handler resource file.

TUNER Sample Program

TV tuner cards allow a desktop PC to receive and display broadcast television signals. TUNER (located in \TOOLKIT\SAMPLES\MM\TUNER), provides an example of how to use the media control interface (MCI) string interface to control a tuner card.

When the application is running, the display window can be sized and a TV channel can be selected. The channel can be selected through up/down buttons or a text entry field displayed at the bottom of the window.

Hardware requirements:

- Computer capable of running OS/2 Warp
- TV tuner card (such as the WinTV Basic)

Software requirements:

- OS/2 Warp
- Multimedia support
- Appropriate drivers for installed tuner card

A TV tuner card is handled as a digital video device by the multimedia subsystem. A typical system has one or more digital video devices, with digitalvideo01 assigned to software motion video. By default this sample opens the digitalvideo02 device. If the tuner card is not assigned to digital video 2 an alternate device ordinal must be provided at the command line with the following format:

```
/d=digitalvideoxx
```

where *xx* is a two digit number, padded on the left with zero.

For example:

```
/d=digitalvideo03
```

Most TV tuner cards, such as the Hauppauge WinTV card, support several video sources. The connector number that corresponds to the tuner video source is dependent on the hardware and might not be the same for each tuner card. The connector number is not changed by this sample. Instead, it uses the default connector number that is set by the Multimedia Setup program.

ULTIEYES Sample Program

ULTIEYES (located in \TOOLKIT\SAMPLES\MM\ULTIEYES) demonstrates the use of non-linear video by displaying segments from a movie clip in response to input from the mouse.

Hardware requirements:

- SVGA (a screen mode of 640x480x256 constitutes a minimum SVGA system)

Software requirements:

- Multimedia support
- Software motion video
- OS/2 Warp

Note: The files for the samples will be installed when the samples are selected, but Workplace Shell objects will not be created for them if the installed operating system is not OS/2 Warp Version 3.

ULIOT I/O Procedure Sample

ULIOT (located in \TOOLKIT\SAMPLES\MM\ULTIMOIO) provides a detailed example of what you need to consider when writing I/O procedures for software motion video file formats. This sample program includes CODEC support and illustrates how to integrate common and file-format-specific code to support multiple I/O procedures.

Open32 Samples

The following Open32 samples are included in the OS/2 Warp Toolkit:

- Browser
- HiWorld
- ToyBox
- DLL Initialization Entry Point (DLLMAIN.C)
- WinMain Wrapper Function (MAIN.C)

Note: Open32 was formerly known as the IBM Developer API Extensions for OS/2.

Browser

The Browser sample enables you to view the lines of text files in a Multiple Document Interface (MDI) environment. The sample provides files containing the program source code. You can open one or more of them, scroll them, change their display fonts, and arrange their windows or icons inside the sample program window.

There are two versions of the Browser sample. The Win32 version, located in the \TOOLKIT\SAMPLES\OPEN32\BROWSER subdirectory, will build without source changes on a Win32 system with the appropriate development tools. (Win32 tools are not provided with the OS/2 Warp Toolkit.)

The OS/2 version of Browser is located in the \TOOLKIT\SAMPLES\OPEN32\BROWSER2 subdirectory.

HiWorld

HiWorld is a simple Windows 32-bit (Win32) application that displays the following text, centered in the client area of the main window:

```
Hello, World
```

There are two versions of the HiWorld sample. The Win32 version, located in the \TOOLKIT\SAMPLES\OPEN32\HIWORLD subdirectory, will build without source changes on a Win32 system with the appropriate development tools. (Win32 tools are not provided with the OS/2 Warp Toolkit.)

The OS/2 version of HiWorld is located in the \TOOLKIT\SAMPLES\OPEN32\HIWORLD2 subdirectory along with a README that briefly describes the steps that were taken to convert the Win32 version of HiWorld to this native OS/2 executable.

ToyBox

ToyBox is a basic Win32 application that uses bit-block transfer (BitBlt) to display a large number of simple bitmaps on the display screen and give them motion. It moves the objects around the client portion of the screen and makes the objects appear to change shape and rotate.

There are two versions of the ToyBox sample. The Win32 version, located in the \TOOLKIT\SAMPLES\OPEN32\TOYBOX subdirectory, will build without source changes on a Win32 system with the appropriate development tools. (Win32 tools are not provided with the OS/2 Warp Toolkit.)

The OS/2 version of ToyBox is located in the \TOOLKIT\SAMPLES\OPEN32\TOYBOX2 subdirectory.

DLL Initialization Entry Point

The DLLMAIN.C file (located in the \TOOLKIT\SAMPLES\OPEN32\DLLENTRY subdirectory) contains the DLL initialization/termination function, which is called when a process gains or loses access to the DLL. The .DEF file used to build the DLL needs to specify INITINSTANCE and TERMINSTANCE; otherwise, this function will be called only for the first process to gain access and the last process to free up the DLL.

This implementation is for IBM VisualAge C++ and assumes that the C Runtime library is being statically linked to the DLL and that the library uses C++ classes.

WinMain Wrapper Function

The MAIN.C file (located in the \TOOLKIT\SAMPLES\OPEN32\WINMAIN subdirectory) is provided as a helper (stub) to demonstrate how to invoke the Windows WinMain function from OS/2.

This is the "main" wrapper for an application based on Open32. It initializes the Alternative Windows Emulator (AWE) environment, calls the WinMain function, and upon completion, calls the WinTerm function to shut down the AWE environment.

Note: To be able to use the Windows WinMain() function, use the OS/2 Warp main() function located in the MAIN.C file. MAIN.C gets compiled and linked with the module containing WinMain() and creates an OS/2 Warp executable file. If you do not use the OS/2 Warp main() function, you will receive a link error stating that there is no starting address for your program.

OpenGL Sample Programs

The OpenGL sample programs are categorized as follows.

Note that installation of OpenGL is required for execution of the OpenGL samples.

- AUXDEMO - Auxiliary Library samples
- GLUTDEMO - Graphics Library Utility Toolkit samples

AUXDEMO Sample Programs

The following Auxiliary Library samples are provided in the OS/2 Warp Toolkit:

- ACCANTI

  This program shows accumulation of jittered objects with an orthographic projection in order to antialias a scene.

- ACCNOT

  This program is basically the same as ACCANTI, but without the jittering to show the same scene aliased.

- ACCPERSP

  This program shows the same scene as the prior ACCANTI and ACCNOT, but with a perspective projection and accumulation of jittered images to antialias the scene.

- ACCUM

  This program is another example of accumulating multiple images in order to antialias a scene.

- AIM

This program calculates the fovy (field of view angle in the Y direction), by using trigonometry, given the size of an object and its size.

- ALPHA

  This program draws several overlapping filled polygons to demonstrate the effect order has on alpha blending results.

- ALPHA3D

  This program demonstrates how to intermix opaque and alpha blended polygons in the same scene, by using glDepthMask(). By pressing the left mouse button, you can toggle the eye position.

- ANTI

  This program draws antialiased lines in RGBA mode.

- ANTIINDX

  This program draws a wireframe icosahedron with antialiased lines, in color index mode.

- ANTIPIDX

  This program draws antialiased points, in color index mode.

- ANTIPOLY

  This program draws filled polygons with antialiased edges. The special GL_SRC_ALPHA_SATURATE blending function is used. By pressing the left mouse button, you can turn the antialiasing on and off.

- ANTIPT

  This program draws antialiased points, in RGBA mode.

- BEZCURVE

  This program uses evaluators to draw a Bezier curve.

- BEZSURF

  This program renders a lighted, filled Bezier surface, using two-dimensional evaluators.

- BEZMESH

  This program renders a wireframe Bezier surface, using two-dimensional evaluators.

- CHECKER

  This program texture maps a checkerboard image onto two rectangles. This program clamps the texture, if the texture coordinates fall outside 0.0 and 1.0.

- CHECKER2

  This program texture maps a checkerboard image onto two rectangles. This program repeats the texture, if the texture coordinates fall outside 0.0 and 1.0.

- CHESS

  This program texture maps a checkerboard image onto two rectangles. The texture coordinates for the rectangles are 0.0 to 3.0.

- CLIP

  This program demonstrates arbitrary clipping planes.

- COLORMAT

  This program will be in ColorMaterial mode after initialization. By pressing the mouse buttons, you can change the color of the diffuse reflection.

- CONE

  This program demonstrates the use of the GL lighting model. A sphere is drawn using a grey material characteristic. A single light source illuminates the object.

- CUBE

  This program draws a 3-D cube, viewed with perspective, stretched along the Y-axis.

- CURVE

  This program uses the Utility Library NURBS routines to draw a one-dimensional NURBS curve.

- DEPTHCUE

  This program draws a wireframe model, which uses intensity (brightness) to give clues to distance. Fog is used to achieve this effect.

- DISK

  This program demonstrates the use of the quadrics Utility Library routines to draw circles and arcs.

- DOF

  This program demonstrates use of the accumulation buffer to create an out-of-focus depth-of-field effect. The teapots are drawn several times into the accumulation buffer. The viewing volume is jittered, except at the focal point, where the viewing volume is at the same position, each time. In this case, the gold teapot remains in focus.

- DOFNOT

  This program demonstrates the same scene as DOF.C, but without use of the accumulation buffer, so that everything is in focus.

- DOUBLE

  This program demonstrates double buffering for flicker-free animation. The left and middle mouse buttons start and stop the spinning motion of the square.

- DRAWF

  This program draws the bitmapped letter F on the screen several times. This demonstrates use of the glBitmap() function.

- FEEDBACK

  This program demonstrates use of OpenGL feedback. First, a lighting environment is set up and a few lines are drawn. Then feedback mode is entered and the same lines are drawn. The results in the feedback buffer are printed.

- FOG

  This program draws five red teapots, each at a different Z-distance from the eye, in different types of fog. By pressing the left mouse button, you can choose between three types of fog: exponential, exponential squared, and linear. In this program, there is a fixed density value, as well as fixed start and end values for the linear fog.

- FOGINDEX

  This program demonstrates fog in color index mode. Three cones are drawn at different Z-values in a linear fog. Thirty-two contiguous colors (from 16 to 47) are loaded with a color ramp.

- FONT

  This program draws some text in a bitmapped font using glBitmap() and other pixel routines. FONT also demonstrates use of display lists.

- LIGHT

  This program demonstrates the use of the OpenGL lighting model. A sphere is drawn using a grey material characteristic. A single light source illuminates the object.

- LINELIST

  This program demonstrates the use of display lists to call different line stipples.

- LINES

  This program demonstrates different line stipples and widths.

- LIST

  This program demonstrates how to make and execute a display list. Note that attributes, such as current color and matrix, are

changed.

- LIST2

  This program demonstrates the use of glGenList() and glPushAttrib(). The matrix and color are restored before the line is drawn.

- MAPLIGHT

  This program demonstrates the use of the GL lighting model. A sphere is drawn using a magenta diffuse reflective and white specular material property. A single light source illuminates the object. This program illustrates lighting in color-map mode.

- MATERIAL

  This program demonstrates the use of the GL lighting model. Several objects are drawn using different material characteristics. A single light source illuminates the objects.

- MIPMAP

  This program demonstrates using mipmaps for texture maps. To overtly show the effect of mipmaps, each mipmap reduction level has a solid colored, contrasting texture image. Thus, the quadrilateral is drawn with several different colors.

- MODEL

  This program demonstrates the use of OpenGL modeling transformations. Four triangles are drawn, each with a different transformation.

- MVLIGHT

  This program demonstrates when to issue lighting and transformation commands to render a model with a light, which is moved by a modeling transformation (rotate or translate). The light position is reset after the modeling transformation is called. The eye position does not change.

  A sphere is drawn using a grey material characteristic. A single light source illuminates the object.

  By pressing the left or middle mouse button, you can alter the modeling transformation (X-rotation) by 30 degrees. The scene is then redrawn with the light in a new position.

- NURBS

  This program shows a NURBS (Non-uniform rational B-splines) surface, shaped like a heart.

- PICKDPTH

  This program demonstrates picking. In rendering mode, three overlapping rectangles are drawn. When you press the left mouse button, you enter selection mode with the picking matrix. Rectangles that are drawn under the cursor position are " picked." Note the depth value range that is returned.

- PICKLINE

  This program demonstrates picking. Press the left mouse button to enter picking mode. You get two hits if you press the mouse, while the cursor is where the lines intersect.

- PICKSQR

  This program demonstrates using multiple names and picking. A 3x3 grid of squares is drawn. When you press the left mouse button, all squares under the cursor position change color.

- PLANE

  This program demonstrates the use of local versus infinite lighting on a flat plane.

- PLANET

  This program shows how to composite modeling transformations to draw translated and rotated models.

  By pressing the Left, Right, Up, or Down Arrow key, you can alter the rotation of the planet around the sun.

- PLANETUP

  In this program, the planets (from PLANET.C) have been rotated so their polar regions are north/south.

  By pressing the Left, Right, Up, or Down Arrow key, you can alter the rotation of the planet around the sun.

- POLYS

  This program demonstrates polygon stippling.

- ROBOT

  This program shows how to composite modeling transformations to draw translated and rotated hierarchical models.

  By pressing any of the arrow keys, you can alter the rotation of the robot arm.

- SCENE

  This program demonstrates the use of the GL lighting model. Objects are drawn using a grey material characteristic. A single light source illuminates the objects.

- SCNBAMB

  This program demonstrates use of a blue ambient light source.

- SCNFLAT

  This program draws lighted objects with flat shading.

- SCNLIGHT

  This program demonstrates the use of a colored (magenta, in this example) light source. Objects are drawn using a grey material characteristic. A single light source illuminates the objects.

- SELECT

  This program is an illustration of the selection mode and name stack, which detect whether objects collide with a viewing volume. First, four triangles and a rectangular box representing a viewing volume are drawn (drawScene routine). The green triangle and yellow triangles appear to lie within the viewing volume, but the red triangle appears to lie outside it. Then, the selection mode is entered (selectObjects routine). Drawing to the screen ceases. To see if any collisions occur, the four triangles are called. In this example, the green triangle causes one hit with the name 1 and the yellow triangles cause one hit with the name 3.

- SIMPLE

  This program draws a white rectangle on a black background.

- SMOOTH

  This program demonstrates smooth shading. A smooth shaded polygon is drawn in a 2-D projection.

- SPHERE

  This program draws a wire frame sphere. It uses glTranslatef() as a viewing transformation.

- STENCIL

  This program draws two rotated tori in a window. A diamond in the center of the window masks out part of the scene. Within this mask, a different model (a sphere) is drawn in a different color.

- STROKE

  This program demonstrates some characters of a stroke (vector) font. The characters are represented by display lists, which are given numbers which correspond to the ASCII values of the characters. Use of glCallLists() is demonstrated.

- SURFACE

  This program draws a NURBS surface in the shape of a symmetrical hill.

- TEA

  This program demonstrates two-sided lighting and compares it with one-sided lighting. Three teapots are drawn, with a clipping plane to expose the interior of the objects.

- TEAAMB

  This program renders three lighted, shaded teapots, with different ambient values.

- TEAPOTS

This program demonstrates several material properties. A single light source illuminates the objects.

- TEXGEN

  This program draws a texture mapped teapot with automatically generated texture coordinates. The texture is rendered as stripes on the teapot.

- TEXSURF

  This program uses evaluators to generate a curved surface and automatically generated texture coordinates.

- TRIM

  This program draws a NURBS surface in the shape of a symmetrical hill, using both a NURBS curve and pwl (piecewise linear) curve to trim part of the surface.

GLUTDEMO Sample Programs

The following Graphics Library Utility Toolkit (GLUT) samples are provided in the OS/2 Warp Toolkit:

- ATLANTIS

  ATLANTIS displays sea-life (whales, sharks, and dolphins) using OpenGL for rendering and the Graphics Library Utility Toolkit for window and user interaction control.

- TEST7

  TEST7 demonstrates window manipulation using the Graphics Library Utility Toolkit.

OS/2 Sample Programs

The OS/2 sample programs are as follows:

- CLOCK - Timer Services Sample
- DLLAPI - Dynamic Link Library Sample
- EAS - Extended Attributes Editor
- HANOI - Multithreaded Sample (Towers of Hanoi)
- NPIPE - Named Pipe Sample
- QUEUES - Interprocess Communication Queue Sample
- SEMAPH - Semaphore Sample
- SORT - Mulithreaded Sample (Sorting Algorithm)
- TIMER0 - High Resolution Timer Sample
- VMM - Virtual Memory Management Sample
- WORMS - Console I/O Sample

CLOCK Sample Program

CLOCK (located in \TOOLKIT\SAMPLES\OS2\TIMESERV) demonstrates how to use and implement window timers and system-resource timers. This sample program displays both an analog and digital clock. To simulate elapsed seconds, the main PM thread repeatedly sets a one-second window timer that updates the current time.

CLOCK features an audible and visual alarm that you can set. When the time expires, the sample makes use of the DOS timer services and notifies the user by sounding an alarm and displaying a message box.

DLLAPI Sample Program

DLLAPI (located in \TOOLKIT\SAMPLES\OS2\DLLAPI) demonstrates how to write and use a dynamic link library (DLL). The sample has a .DLL file and an executable (.EXE) file. The .DLL provides the 32-bit API function that is called by the .EXE file.

The .DLL uses protected memory on its shared data, and exception management to validate the pointer parameters for a 32-bit API function. The .EXE file demonstrates how to handle a divide-by-zero exception, and calls the function with invalid pointer parameters, followed by a call with valid pointer parameters.

EAS Sample Program

EAS (located in \TOOLKIT\SAMPLES\OS2\EAEDIT) demonstrates a multithreaded application that retrieves, modifies, or sorts files by their extended attribute value. Included in this sample program are PM procedures for dialog boxes and a standard client window. The sample lets you select an extended attribute file name from a list, or enter a new name in an entry field. You can select the extended attribute type from a table.

HANOI Sample Program

HANOI (located in \TOOLKIT\SAMPLES\OS2\HANOI) demonstrates a multithreaded application with the familiar "towers of Hanoi" puzzle. When the sample program is started, you see three poles (A, B, and C). Initially, pole A has on it a stack of disks starting with the largest disks on the bottom and succeeding smaller disks on the top. The main thread handles the PM interface and lets the you start or stop the Hanoi routine. It also lets you reset the number of working disks. The second thread is created when the **Start** item is selected from the **Options** menu. This thread starts the recursive execution of the Hanoi algorithm, runs in the background, and moves and paints the disks. All disks end up on pole C.

NPIPE Sample Program

NPIPE (located in \TOOLKIT\SAMPLES\OS2\NPIPE) demonstrates two-way communication between two unrelated processes using named pipe functions. This sample program implements the game of TicTacToe with two executable files:

- CLINPIPE.EXE (the client)

  The client is the user. For example, the client will:

  - Connect to the server and acknowledge successful connection (START_MSG).
  - Notify the server through a pipe when it wishes to begin play (YOU_FIRST or CLIENT_MOVE).
  - Notify the server when it wishes to quit (CLIENT_QUIT).
  - Send the server a valid move when requested by the server (CLIENT_MOVE).

- SVRNPIPE.EXE (the server)

  The server is the computer. For example, the server will:

  - Connect a pipe to the client through which play will be executed (START_MSG) upon the initial request of a client to play.

- Play with many clients simultaneously.
- Notify the client of the server's move, and request a valid move from the client (SERVER_MOVE).
- Notify the client of game-end (WIN_SERVER, WIN_CLIENT, WIN_DRAW).

QUEUES Sample Program

QUEUES (located in \TOOLKIT\SAMPLES\OS2\QUEUES) demonstrates interprocess communications (IPC) using the 32-bit queue component. It consists of two executable programs:

- SVRQUEUE.EXE

  Creates an IPC queue; a named, shared-memory buffer for queue elements; and a shared, named, mutex (mutual exclusive) semaphore. After initializing the queue, SVRQUEUE starts a thread to read from the queue, prints the contents of the messages read from the queue, and terminates at the user's request.

- CLIQUEUE.EXE

  Opens the queue and accesses the shared-memory element buffer and mutex semaphore, and starts a thread to write to the queue. CLIQUEUE requests a string of data from the user, allocates a shared-memory element from the buffer, puts the string in the shared-memory element, and then uses an event semaphore to direct the thread to write the element to the queue. CLIQUEUE terminates at the user's request.

SEMAPH Sample Program

SEMAPH (located in \TOOLKIT\SAMPLES\OS2\SEMAPH) demonstrates the use of mutex and event semaphores. In the sample, several threads share access to the same resource.

A mutex semaphore is used to guarantee that only one thread has access to the resource at a time. A mutex semaphore is used to check for a stop event or for a user signal to give up the resource.

An event semaphore is used to signal the thread to give up the resource. An event semaphore can be posted by the user, or run in auto mode, in which case the event semaphore will be posted at fixed time intervals.

Each thread can be displayed as a square of a unique color. Similarly, the resource can be displayed as a rectangle; its color is that of the first thread that owns it.

SORT Sample Program

SORT (located in \TOOLKIT\SAMPLES\OS2\SORT) demonstrates the use of multiple threads by performing multiple sorts at the same time. Each sorting algorithm runs from a separate thread. The main thread is used to handle the main window's messages, while the routine that updates the display is run from another thread.

TIMER0 Sample Program

TIMER0 (located in \TOOLKIT\SAMPLES\OS2\TIMER0) creates four threads. Each thread does the following:

1. Sets its priority to time-critical.

2. Sleeps for *n* milliseconds, where *n* is equal to 500 times the number of the thread (for example, thread #1 sleeps 500ms, thread #2 sleeps 1000ms, and so on).

3. Wakes up and increments a counter.

To start the TIMER0 sample from an OS/2 command line, type:

```
TIMER0
```

While the four threads are running, the main thread (the one that created the other four) is in an infinite loop.

Note: To break out of the loop, press any key.

This loop displays the values of each of the four counters incremented by the four threads. For example:

```
TIMER0$ opened.  File Handle is 3
Counter: 00008703  1:00000017 2:00000008 3:00000005 4:00000004
```

The counter is the value of the millisecond counter in timer TIMER0 device driver. This value gets incremented every millisecond and is very accurate. The other four values are the four counters and these values are incremented by each thread. The accuracy is dependent on the thread contention and other things that are going on in the system. Note that #2 is approximately one-half of #1, #3 is approximately one-third of #1, and #4 is approximately one-fourth of #1.

VMM Sample Program

VMM (located in \TOOLKIT\SAMPLES\OS2\VMM) demonstrates the use of virtual memory by using new memory-management functions to allocate and set the attributes of memory. Users can read or write data into memory and reset the attributes using a dialog box. The memory manager protects or opens the virtual memory to read or write operations according to the different attributes of each memory block. To free memory, the user enters the address of the memory.

WORMS Sample Program

WORMS (located in \TOOLKIT\SAMPLES\OS2\CONSOLIO) demonstrates how to call video (Vio), keyboard (Kbd), and mouse (Mou) 16-bit function from a 32-bit code segment. This sample program displays earthworms aimlessly moving about the screen. Each worm is a separate thread with a unique color combination and movement pattern. When one worm encounters another worm, the color attribute of the worm is set to red. You can add or delete worms using the keyboard or mouse.

Presentation Manager Sample Programs

The PM sample programs are as follows:

- CLIPBRD - Clipboard Sample
- DIALOG - Introductory Dialog Box Sample
- DRAGDROP - Direct Manipulation (Dragdrop) Sample
- GRAPHIC - Non-retained Graphics Sample
- HELLO - Standard Windows Sample
- IMAGE32 - PM 16-bit to 32-bit Porting Sample
- IPF - Information Presentation Facility Sample
- JIGSAW - Bitmap Manipulation Sample
- PALETTE - Palette Sample
- PRINT - Printer Sample Program

- STYLE - PM Controls Sample
- TEMPLATE - Application Template

CLIPBRD Sample Program

CLIPBRD (located in \TOOLKIT\SAMPLES\PM\CLIPBRD) demonstrates how to provide a PM interface to the clipboard. Initially, this sample program displays a standard window with a bitmap. You can cut and paste rectangular images in this window, using the system clipboard as an intermediate storage area.

DIALOG Sample Program

DIALOG (located in \TOOLKIT\SAMPLES\PM\DIALOG) demonstrates how to associate a dialog box with a standard window. The dialog box is defined as a dialog template in a resource file.

This sample program also demonstrates how to implement entry fields, push buttons, and message boxes. Message boxes are displayed only for error conditions.

DRAGDROP Sample Program

DRAGDROP (located in \TOOLKIT\SAMPLES\PM\DRAGDROP) demonstrates how to move files between directories by using the drag and drop operation of direct manipulation. This sample program creates a drop-down list box that contains a scrollable file name list of the current directory.

To change the current directory, select the **Window** option, type in the directory name, and press Enter. To change the current directory to one higher in the directory tree, select **File** from the menu bar, and then **Open**. The Select subdirectory window is displayed. Type in the name of the subdirectory, and then select OK.

DRAGDROP must be started twice, so that there are two running instances of the sample. Then, using a mouse, you can:

- Display a directory file list in the first sample.
- Select a file name from the second sample.
- Drag the file name (using the right mouse button) to the directory in the first sample.
- Drop the file name into the directory in the first sample.

The file is now moved to the chosen directory of the first sample.

GRAPHIC Sample Program

GRAPHIC (located in \TOOLKIT\SAMPLES\PM\GRAPHICS) demonstrates how to use default viewing transformation functions of the PM. It also demonstrates how to use an asynchronous drawing thread. This sample program lets you load metafiles using a dialog box. The dialog box contains a Help push button. When the Help push button is activated, it provides instructions on loading a metafile from another directory. You also can print a metafile or graphic circle.

HELLO Sample Program

HELLO (located in \TOOLKIT\SAMPLES\PM\STDWND) demonstrates how to create and display a standard window that conforms to the *Common User Access* requirements.

This sample program also demonstrates how to use resources defined in a script file. Initially, HELLO displays a standard window with the text "Hello." The **Options** conditional-cascaded menu contains three items. Each item paints a different text string in the window.

This sample also shows how to override the Ctrl+C key combination.

IMAGE32 Sample Program

IMAGE32 (located in \TOOLKIT\SAMPLES\PM\PORTING) demonstrates how to migrate an existing OS/2 16-bit application to a 32-bit application.

This sample also demonstrates how to display an image using the GpiImage function. The image data comes from a file that you must select using the standard **File Open** menu item.

IPF Sample Program

IPF (located in \TOOLKIT\SAMPLES\PM\IPF) demonstrates how to use the IPF to create an online document. This sample program features customized windows that display text, graphics, and animation.

Two files are associated with this sample:

- IPF online document (INF)

  The .INF file is the compiled IPF document. The source contains tagging that defines different types of windows. Tags that control the format and display of text also are included in this file.

- OS/2 dynamic link library (DLL)

  The .DLL file is the compiled OS/2 C language source for the code object that is called when the .INF file is read at run time. A series of bitmaps used for animation are included in the .DLL.

JIGSAW Sample Program

JIGSAW (located in \TOOLKIT\SAMPLES\PM\BMPSAMP) demonstrates the use of bitmaps in a graphics application. This sample provides a jigsaw puzzle based on the decomposition of an arbitrary bitmap loaded from a file. You can jumble the pieces, and then drag them with a mouse. The image can be made smaller, larger, scrolled horizontally, or scrolled vertically.

This sample program also demonstrates how to call the Information Presentation Facility help hook, to create an instance and associate the instance with the active application window.

PALETTE Sample Program

PALETTE (located in \TOOLKIT\SAMPLES\PM\PALETTE) demonstrates 32-bit graphics functions including:

- Creating a window using a custom palette and animation.
- Using menus with switches, and modifying the menu text.
- Using multiple threads and semaphores in the Presentation Manager environment.
- Displaying graphics on the screen using outline fonts and clip paths.
- Online help.

When started, PALETTE displays a standard window with a large OS/2 logo in the foreground. You have the ability to change the OS/2 logo to the IBM logo. If you resize the window, the logo is scaled and redrawn to fit the new window size. You can also control the animation speed from the PALETTE menu.

The animation is performed by:

- Creating a clip path that represents the outline of the logo characters (which are displayed using an outline font).
- Setting the clip path to the presentation space.
- Drawing a series of lines to the presentation space.

Each line is drawn with an incremental color index. Palette animation is performed using the 32-bit GpiAnimatePalette function.

In order for PALETTE to remain responsive to system and user messages, no animation is performed on the main window procedure thread. A second thread is created from which all animation is performed.

Hardware requirements:

- XGA adapter
- 1MG of RAM
- 32-bit graphics engine

PRINT Sample Program

PRINT (located in \TOOLKIT\SAMPLES\PM\PRINT) demonstrates how to display and print text, metafiles, and bitmaps. It also demonstrates how to:

- Query and display system printer configurations
- Interact with printer drivers to change job properties
- Query and display available printer and screen fonts
- Query and display printer forms and setup margins
- Selectively print part or all of a document on an asynchronous thread.

STYLE Sample Program

STYLE (located in \TOOLKIT\SAMPLES\PM\CONTROLS) demonstrates a PM application that conforms with the Common User Access requirements and implements the following controls:

- Container
- Notebook
- Slider
- Spin button
- Value set

This sample program also demonstrates secondary windows, such as dialogs and message boxes. The program lets you edit and save text files. The source for online help, in IPF format, is also provided.

STYLE also demonstrates the detection of a font that does not conform to the International Standards Organization (ISO 9241). When you are running the sample on an ISO-compliant monitor and select a non-compliant font in the standard font dialog, a message box is displayed.

The code in STYLE is structured so that the addition of a new function is handled in an efficient manner. For example, to add a new command to an existing menu, you need only add the command to the resource file, and then add the appropriate message-processing routines to the STY_USER.C file.

TEMPLATE Sample Program

TEMPLATE (located in \TOOLKIT\SAMPLES\PM\TEMPLATE) demonstrates the structure common to all PM applications. This sample program shows how to structure an application that has more than one source file. It includes the initialization file (which is used and then discarded), the resident code, and the non-resident code that is loaded only when needed.

TEMPLATE also demonstrates how to:

- Create a standard window
- Load resources from a resource file
- Create a dialog box and a button control
- Display a message box
- Open a file
- Close a file
- Print text
- Paint a window
- Process a message from a menu
- Run a thread in the background
- Exit a process.

Several online help files are also provided in IPF format.

Problem Determination Sample Programs

The Problem Determination sample programs are categorized as follows:

- DMI - Desktop Management Interface Samples
- ERRLOG - Logging Framework Samples
- FFST - First Failure Support Technology Sample
- TRACE - Trace Sample

DMI

The following desktop management interface (DMI) samples are included with the OS/2 Warp Toolkit. They are used in conjunction with the DMI MIF Browser, which is provided with the OS/2 Warp Version 4 or later operating system.

- DMI Overlay (DLL) Componentry Sample
- DMI Direct Interface Componentry Sample

DMI Overlay (DLL) Componentry Sample

The DMI Overlay Componentry sample (NAMES.C) is located in the \TOOLKIT\SAMPLES\SYSMGT\DMI\NAMES subdirectory. To use NAMES, do the following:

1. Install the NAMES.MIF file into the DMI database.

a. Open the Problem Determination Tools folder (located in the OS/2 System folder).

b. Start the DMI MIF Browser.

c. Select **Install new** from the Components menu.

d. Select NAMES.MIF.

e. Do not close the DMI MIF Browser.

2. Copy the NAMES.DLL to a directory in the LIBPATH.

3. A component called DMI Service Layers will be created in the DMI MIF browser. Expand the component and double-click on one of the groups to see information about the operating system, company name, and the name of the DMI service layer. This information is coming from the NAMES.DLL file.

DMI Direct Interface Componentry Sample

The DMI Direct Interface Componentry sample (NAMEDIR.C) is located in the \TOOLKIT\SAMPLES\SYSMGT\DMI\NAMEDIR subdirectory.

To use NAMEDIR, do the following:

1. Copy the NAMEDIR.MIF file to the \DMIPATH directory.

2. Open the Problem Determination Tools folder (located in the OS/2 System folder).

3. Start the DMI MIF Browser.

4. From an OS/2 session, run the NAMEDIR.EXE program. The following two things will happen:

   - A new component will be added to the DMI database and will be displayed in the DMI MIF Browser. The name of the component will be DMI Service Layers - Direct Interface (DI) version.

   - The OS/2 session will display the following messages:

```
Preparing to install 'NAMEDIR.MIF'
Installation successful ComponentID=6.
Preparing to register component id 6
I got started
```

5. Expand the new component and double-click on one of the groups to see information about the operating system, company name, and the name of the DMI service layer. This information is coming from the NAMEDIR.EXE file.

ERRLOG

The following samples demonstrate use of the logging functions.

- ERRLOG1
- ERRLOG2

It is helpful to run these sample files using a debugger so that you can step through the code to see the function the API is actually providing.

ERRLOG1

ERRLOG1 (located in \TOOLKIT\SAMPLES\SYSMGT\ERRLOG\ERRLOG1) demonstrates how to use the following functions to open a file, read an entry from the log, format that entry, and then close the log.

LogOpenFile
LogReadEntry
LogFormatEntry
LogCloseFile

The program displays a message indicating whether each function call was successful or not.

If the error log is empty, the program will end after the LogReadEntry function call with a return code of 1750 indicating the end of the file was reached without finding the desired record. To have the program run successfully, run the FFSTProbe sample program (PROBE.EXE) to put an entry in the log. PROBE.EXE is located in the \TOOLKIT\SAMPLES\SYSMGT\FFST\PROBE subdirectory.

ERRLOG2

ERRLOG2 (located in \TOOLKIT\SAMPLES\SYSMGT\ERRLOG\ERRLOG2) demonstrates how to use the following functions to open event notification, wait on an error log event, change an event filter, and then close event notification.

LogOpenEventNotification
LogWaitEvent
LogChangeEventFilter
LogCloseEventNotification

The program displays a message after each error logging function call indicating whether the call was successful or not. Because this program is using the event notification functions, it will pause twice as it's waiting for an error to be logged. At this time it will display a message indicating that it is waiting for an event.

When event notification is started a filter is not specified. Therefore, on the first use of LogWaitEvent, any record placed in the log will cause the notification to occur. The LogChangeEventFilter function is then used to filter on records where the vendor tag field contains IBM or the severity of the error is less than, or equal, to 2. In either case, the FFSTProbe sample program (PROBE.EXE) will cause the notification to occur.

To run ERRLOG2, do the following:

1. At an OS/2 command line, type:

   ```
   ERRL0G2
   ```

   It will run to the point where it displays a message stating that it is waiting for an event.

2. Open another OS/2 window and start the FFSTProbe sample program by changing to the \TOOLKIT\SAMPLES\SYSMGT\FFST\PROBE subdirectory and typing:

   ```
   PR0BE
   ```

   The ERRLOG2 program will display messages stating that it has received an event, that it's changing the event filter, and that it is waiting for an event.

3. Open another OS/2 window and start the FFSTProbe sample program again.

   The ERRLOG2 program will again display a message stating that it has received an event, followed by a message stating that it is closing event notification.

FFST

PROBE (located in \TOOLKIT\SAMPLES\SYSMGT\FFST\PROBE) shows how to use the FFSTProbe function. You should use the

FFSTProbe function with a "wrapper" function. An example of a wrapper function is included in this sample code. You can add or delete parameters sent to the wrapper function based on your own needs.

To run the PROBE sample, do the following:

1. Copy PROBE.MSG and PROBE.REP into the \OS2\SYSTEM directory. PROBEMSG.TXT and PROBEREP.TXT are the source files used to create the message and template repository files.
2. Install the MIF file into the DMI database.
    a. Open the Problem Determination Tools folder (located in the OS/2 System folder).
    b. Start the DMI MIF Browser.
    c. Select **Install new** from the Components menu.
    d. Select PROBE.MIF.
    e. Do not close the DMI MIF Browser.
3. At an OS/2 command line type:

    ```
    PROBE
    ```

After the PROBE sample runs, use the System Error Logger (in the Problem Determination Tools folder) to view the two probe entries, 1111 and 2222. Double-click on the entries to view detailed information.

TRACE

TRACEEXP (located in \TOOLKIT\SAMPLES\SYSMGT\TRACE) shows how to create a trace event in the system-event trace buffer. The Trace Create Entry Request packet is initialized with the major code, minor code, length of data to be collected (if any), and starting address of the data (or NULL). The size of the packet and the trace release number must also be supplied.

The call to TraceCreateEntry() will place this data into the system-event trace buffer if the major and minor codes were turned on by the user. If trace is not enabled, or if the major and minor code combination is not turned on, the call to TraceCreateEntry() performs no action.

Use the Trace Formatter, provided with the OS/2 Warp Version 4 or later operating system, to look at the system-event trace buffer.

For more information on using this sample, see Starting TRACEEXP.

Starting TRACEEXP

To run the TRACEEXP sample, you need one or both of the following lines in your CONFIG.SYS:

```
trace=on
tracebuf=4
```

Be sure to restart your system if you make changes to CONFIG.SYS.

Turn on trace for the major code in this example by typing the following:

```
trace on 123
```

To run the sample, type:

```
traceexp
```

To turn the trace off, type:

```
trace off
```

To see the captured trace point, type:

```
tracefmt
```

This should show an entry with major code 123.

REXX Sample Programs

The REXX sample programs are as follows:

- CALLREXX
- DEVINFO
- PMREXX
- REXXCALC
- REXXUTIL
- RXMACDLL
- REXX Animal SOM Class Sample
- REXX Workplace Shell Class Sample
- REXX Workplace Shell Folder Sample
- REXX Complex Number Sample
- REXX Concurrency Sample
- REXX Directives Sample
- REXX Display Month Array Sample
- REXX Factorial Sample
- REXX Guard and Reply Sample
- REXX Pipe Sample
- REXX Query Date Sample
- REXX Query Time Sample
- REXX Semaphore Class Sample
- REXX Stack Sample
- REXX Start and Guard Sample

CALLREXX Sample Program

CALLREXX demonstrates how a C-language application calls a REXX application. To run the REXX application BACKWARD.FNC, CALLREXX.C issues RexxStart. RexxStart calls the REXX interpreter and passes it a parameter string. BACKWARD.FNC returns a result string to the C-language application.

DEVINFO Sample Program

DEVINFO issues DosDevConfig and returns the data in a collection of compound variables when all available items are requested or a single variable when only one item is requested. This is a REXX subcommand handler and variable pool example. This sample can be run in an OS/2 full-screen session, an OS/2 text-window session, or in PMREXX.

PMREXX Sample Program

PMREXX provides a PM window in which you can display the output from a REXX procedure or from any programs called by the REXX procedure. The window has an entry field into which you can type.

REXXCALC Sample Program

REXXCALC illustrates the steps required to develop enhanced applications.

REXXUTIL Sample Program

REXXUTIL demonstrates a set of external functions packaged in a dynamic link library, including:

- Use of OS/2 system functions in REXX external functions
- Techniques for passing large amounts of data to a REXX program using REXX compound variables as arrays.

REXXUTIL runs on 32-bit OS/2. It can be run in an OS/2 full-screen session, an OS/2 window session, or in PMREXX.

RXMACDLL Sample Program

RXMACDLL demonstrates the macrospace interface with the two following C-language programs and that are compiled into two separate dynamic link libraries (DLLs):

- MACRO.C

  Contains REXX external functions, which perform REXX macrospace operations.

- RXNLSINF.C

  Contains a REXX external function that provides information related to national language support (for example, as a currency symbol and separator).

RXMACDLL.CMD uses MACRO.DLL to load NLSMONEY.CMD into the macrospace and calls NLSMONEY.CMD several times to format currency amounts. NLSMONEY.CMD formats the amounts according to the specifications provided by RXNLSINF.DLL.

RXMACDLL can be run in an OS/2 full-screen session, an OS/2 window session, or in PMREXX.

REXX Animal SOM Class Sample

The sample code in the \TOOLKIT\SAMPLES\REXX\SOM\ANIMAL directory illustrates the use of a simple metaclass to provide customized constructors and to perform basic instance tracking.

For this sample to function properly, Object REXX must be installed.

REXX Workplace Shell Class Sample

The WPSORXCL.CMD sample in the \TOOLKIT\SAMPLES\REXX\WPS directory demonstrates how to use the OS/2 Workplace Shell folders to represent Object REXX classes. This is presented in a tree view where an Object REXX class is a folder and subclasses are subfolders. The top folder title is updated as classes are added for a progress indicator. Methods in a class can be represented as abstract WPS objects; but the default is not to create them because it is quite time consuming.

For this sample to function properly, Object REXX must be installed.

The REXX Workplace Shell samples also require the REXX classes first be registered with the Workplace Shell. To register REXX classes with the Workplace Shell, type the following at an OS/2 command prompt:

```
WPSINST
```

The WPSINST.CMD program can be used at any time to query REXX classes and to register and deregister them. To deregister the REXX classes, type the following at an OS/2 command prompt:

```
WPSINST -
```

REXX Workplace Shell Folder Sample

The WPSORXFL.CMD sample in the \TOOLKIT\SAMPLES\REXX\WPS directory demonstrates useful routines for folders. The first routine will take a WPS folder as input and return an array of SOM objects; one for each object in that folder. The second routine will take an array of WPS objects and a string; returning an array of objects that match the string. The sample also uses three methods to iterate over arrays as iteration examples.

For this sample to function properly, Object REXX must be installed.

The REXX Workplace Shell samples also require the REXX classes first be registered with the Workplace Shell. To register REXX classes with the Workplace Shell, type the following at an OS/2 command prompt:

```
WPSINST
```

The WPSINST.CMD program can be used at any time to query REXX classes and to register and deregister them. To deregister the REXX classes, type the following at an OS/2 command prompt:

```
WPSINST -
```

REXX Complex Number Sample

The COMPLEX.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates how to create a complex number class using the ::class and ::method directives. Also shown is an example of subclassing the complex number class (the vector subclass). Finally, the stringlike class demonstrates the use of a mixin to provide some string behavior to the complex number class.

The USECOMP.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates the use of the ::requires directive to use the complex number class defined in COMPLEX.CMD.

For this sample to function properly, Object REXX must be installed.

REXX Concurrency Sample

The CCREPLY.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates how to use reply to run two methods at the same time.

For this sample to function properly, Object REXX must be installed.

REXX Directives Sample

The GUESS.CMD sample in the \TOOLKIT\SAMPLES\REXX directory creates a simple node class and uses it to create a logic tree. The logic tree is filled in by playing a simple guessing game.

For this sample to function properly, Object REXX must be installed.

REXX Display Month Array Sample

The MONTH.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates the use of arrays to replace stems. This sample displays the days of the month.

For this sample to function properly, Object REXX must be installed.

REXX Factorial Sample

The FACTOR.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates a way to define a factorial class using the subclass method and the *.methods* environment symbol.

For this sample to function properly, Object REXX must be installed.

REXX Guard and Reply Sample

The GREPLY.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates the difference between Guarded and UnGuarded methods. In the first example, the method is guarded, so it does not begin execution until the final result is tallied. In the second example, the method is unguarded so it can begin execution while method sum_up is still looping. In fact, unguarded_get often runs immediately after the Reply, so using a guard instruction ensures sum_up runs for a bit before unguarded_get returns with the intermediate sum.

For this sample to function properly, Object REXX must be installed.

REXX Pipe Sample

The PIPE.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates the use of the ::class and ::method directives to create a simple implementation of a CMS-like pipeline function.

The USEPIPE.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates how to use the pipes implemented in the PIPE.CMD sample.

For this sample to function properly, Object REXX must be installed.

REXX Query Date Sample

The QDATE.CMD sample in the \TOOLKIT\SAMPLES\REXX directory displays the current date and moon phase for that date in English.

REXX Query Time Sample

The QTIME.CMD sample in the \TOOLKIT\SAMPLES\REXX directory displays the current time in English.

REXX Semaphore Class Sample

The SEMCLS.CMD sample in the \TOOLKIT\SAMPLES\REXX directory implements a semaphore class in Object REXX. The class is defined to the Global OREXX environment.

For this sample to function properly, Object REXX must be installed.

REXX Stack Sample

The STACK.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates how to implement a stack class using the ::class and ::method directives. This sample also includes a short example of the use of a stack.

For this sample to function properly, Object REXX must be installed.

REXX Start and Guard Sample

The KTGUARD.CMD sample in the \TOOLKIT\SAMPLES\REXX directory demonstrates the use of the start method and the guard instruction to control execution of several programs. In this sample, the programs are controlled by one "guarded" variable.

For this sample to function properly, Object REXX must be installed.

SOM Sample Programs

Note: The ANIMALS and TP (text processing) sample programs are not included in the OS/2 Warp Toolkit. Refer to the SOMobjects Developer Toolkit for accessing them and obtaining information on them.

TCP/IP Sample Programs

The TCP/IP sample programs are categorized as follows:

- ACCXRECV (accept_and_recv function)
- ECHOSAMP
- HPS
- OS2IOCTL
- R0LIB32
- RCOPY
- RPC
- RPCGEN
- SAMPDLL
- SENDFILE
- SOCKET
- SYSCTL

ACCXRECV Sample Program

The accept_and_recv function call is explained in this sample program. The server opens a socket to which it binds and then does a listen on that socket. It starts the number of specified threads and does accept_and_recv on each of them. The client opens a socket, connects to the server and sends the data. As the timeout option is set (by using setsockeopt), whenever the client connects but doesn't send any data, the server times out.

ECHOSAMP Sample Program

This is a client/server application that is implemented using the Echo protocol. Whatever the echo client sends to the server, the server sends it back to the client.

HPS Sample Program

This is a client/server application that is implemented for exploiting High Performance Send. The server opens a socket to which the client connects and passes data using HPS. The server responds to the client using HPS.

OS2IOCTL Sample Program

This sample program issues SIOSTATRT for getting the routing entries from the routing table, which is displayed on the screen, and issues SIOCSSYN to set the SYN attack prevention frunction flag to on.

R0LIB32 Sample Program

The client sends 1k of data (all 'a'). Then, the server modifies the data (changes all 'a' to 'b') and sends the modified data to the client. The client prints the data received from the server. The client sends a string to the client using sendmsg() call, and the server receives it (by recvmsg() call) and sends another string to the client. The client receives the string and prints it. The client program is a ring3 program. The server program accesses the socket calls through r0lib32.

RCOPY Sample Program

RCOPY is located in \TOOLKIT\SAMPLES\TCPIPTK\RCOPY). It makes use of the FTP API, FTPAPI.DLL, to transfer a directory and all of its subdirectories between a client and FTP server. RCOPY does not read the user ID or password from the NETRC file, so they must be specified when required.

The following explains how to build and run the RCOPY samples.

1. To compile and link the sample, type:

   ```
   nmake -f vacpp.mak   (to build using the VisualAge C/C++ compiler)
   ```

2. To run rcopy.exe:

   - Command syntax:

     ```
     RCOPY put|get ascii|binary [-l<local path>] -h<host>
           -u<userid> -p<password> [-r<rem path>] [-a<account>]
     ```

     where:

put	Transfers from local host to remote host (default)
get	Transfers from remote host to local host
binary	Binary transfer mode (default)
ASCII	ASCII transfer mode

   - Example:

     ```
     rcopy put -ld:\tmp -holeg -uyozzo -pyozzops -rc:\newtmp
     ```

     where:

d:\tmp	Is a local path to transfer
oleg	Is a remote host (FTP server)
yozzo	Is a remote user ID
yozzops	Is a remote password

RPC Sample Programs

RPC consists of three C programs located in \TOOLKIT\SAMPLES\TCPIPTK\RPC:

- RAWEX.C provides an example of the RAW client/server usage. Portmapper must be running.
- GENESEND.C sends an integer to the remote host (GENESEND) and receives the integer back at the local host. Portmapper and Remote server (GENESEND) must be running.

- GENESERV.C is a generic server. It receives an integer or float and returns them, respectively. Portmapper must be running.

The following explains how to build and run the RPC samples.

1. To compile and link the sample program, type:

   ```
   nmake  -f rpcsamp.mak
   ```

2. To start Portmapper, type:

   ```
   start portmap
   ```

3. To start the geneserv.exe server, type:

   ```
   start geneserv
   ```

4. To run genesend.exe, type:

   ```
   genesend hostname integer
   ```

   For example:

   ```
   genesend charm 1000
   ```

5. To run rawex.exe, type:

   ```
   rawex integer
   ```

   For example:

   ```
   rawex 1000
   ```

RPCGEN Sample Programs

RPCGEN consists of three user-written files located in \TOOLKIT\SAMPLES\TCPIPTK\RPCGEN):

- RG.X defines the remote procedure characteristics
- RGUS.C demonstrates an RPC server program
- RGUC.C demonstrates an RPC client program

The following explains how to build and run the RPCGEN samples.

1. Perform the following:

   ```
   rpcgen rg.x
   ```

   This generates rg.h, rg.i, rgs.c, rgc.c, and rgx.c.

2. From rgx.c, add the include headers to rgus.c and rguc.c if they do not already exist

3. Run rpcgsamp.cmd to compile rgus.c, rguc.c, rgs.c, and rgc.c

   Link rgs.obj and rgus.obj to generate rgs.exe

Link rgc.obj and rguc.obj to generate rgc.exe

4. Start portmap.exe and rgs.exe on the same machine:

```
start portmap
start rgs
```

5. Start rgc.exe by typing:

```
rgc hostname "user_input string"
```

SAMPDLL Sample Programs

SAMPDLL (located in \TOOLKIT\SAMPLES\TCPIPTK\SAMPDLL) demonstrates the use of a DLL.

MYPROG (located in \TOOLKIT\SAMPLES\TCPIPTK\SAMPDLL) demonstrates building a DLL.

SENDFILE Sample Program

This is a TCP client/server application that is used for explaining the send_file (). The client sends a file to a server.

SOCKET Sample Programs

SOCKET consists of three client/server samples and SELECTS.C located in \TOOLKIT\SAMPLES\TCPIPTK\SOCKET.

SELECTS.C is a simple TCP and UDP server that uses the BSD select() call. MSGC.C/MSGS.C, TCPC.C/TCPS.C, and UDPC.C/UDPS.C are sample C socket client/server programs demonstrating network programming with OS/2 sockets.

The following explains how to build and run the SOCKET samples.

1. To compile and link the sample programs, type:

```
nmake -f vacpp.mak  (to build using the VisualAge C/C++ compiler)
```

2. Running udps.exe and udpc.exe:

   a. To start the udps server, type:

   ```
   start udps
   ```

   An additional session is created and the following message is displayed:

   Port assigned is 1028

   b. To run the udpc client, type:

   ```
   udpc 9.67.60.10 1028
   ```

   where:

9.67.60.10	Is the IP address of the machine where the udps server is running
1028	Is the port assigned by the udps server

After running the udpc client, you will see the following output on the server session:

```
Received message Hello ..............
```

3. Running tcps.exe and tcpc.exe

   a. To start the tcps server, type:

   ```
   start tcps 2000
   ```

   where 2000 is the port number assigned to the tcps server.

   b. To run the tcpc client, type:

   ```
   tcpc charm 2000
   ```

   where:

charm	Is the hostname of the machine where the tcps server is running
2000	Is the port assigned to the tcps server

4. Running selects.exe

   The selects server has the same function as the tcps and udps servers.

   a. To start the selects server, type:

   ```
   start selects 2000
   ```

   where 2000 is the port number assigned to the selects server.

   b. To run the tcpc client, type:

   ```
   tcpc charm 2000
   ```

   where:

charm	Is the hostname of the machine where the selects server is running
2000	Is the port assigned to the selects server

   c. To run the udpc client, type:

   ```
   udpc 9.67.60.10 2000
   ```

   where:

9.67.60.10	Is the IP address of the machine where the selects server is running
2000	Is the port assigned by the selects server

   After running the udpc client, you will see the following output on the server session:

   ```
   Received message Hello ..............
   ```

5. Running msgs.exe and msgc.exe

   msgs.c and msgc.c are the same program as udps.c and udpc.c, except they are written using recvmsg() and sendmsg() calls.

   a. To start the msgs server, type:

```
start msgs
```

An additional session is created and the following message appears:

```
Port assigned is 1028
```

b. To run the msgc client, type:

```
msgc 9.67.60.10 1028
```

where:

9.67.60.10	Is the IP address of the machine where the msgs server is running
1028	Is the port assigned by the msgs server

SYSCTL Sample Program

The sysctl() call accesses and modifies system-wide parameter values for the entire TCP/IP stack. This sample program sets the ipgate to 1 and then back to 0.

Workplace Shell Sample Programs

The Workplace Shell sample programs are as follows:

- BROWSE
- CAR
- CARPP
- TEXTFLDR
- WPSTUTOR
- WSFILE

Note: All Workplace Shell samples require SOM 2.1 in order to execute properly.

BROWSE Sample Program

BROWSE displays file system objects in a PM window in either text or hexidecimal format. Hexidecimal is the default format. To view a file, you can either drop a file system object on the Workplace Shell ASCII/Hex File Browser or double-click on the Workplace Shell ASCII/Hex File Browser and enter the name of a file to view.

To choose the format for viewing an object, click on the Workplace Shell ASCII/Hex File Browser and select **Open Text View** or **Open Hex View** from the **Open** menu.

The default view format can be changed in the Settings notebook of the Workplace Shell ASCII/Hex File Browser by doing the following:

1. Select the **Menu** tab.
2. Select **~Open**from the Available menus: list box.
3. Select **Settings**.

4. Select **Open ~Hex View**or **Open ~Text View**from the Default action list box.

5. Select **OK**.

CAR Sample Program

CAR demonstrates how to create a Workplace Shell object using basic object-oriented programming techniques and the SOM, including:

- Initializing an object
- Adding settings pages to an object
- Saving and restoring the state of an object
- Modifying an object's context menus (adding and deleting menu items)
- Querying object class data
- Processing context menu items
- Implementing settings page dialog processing
- Handling exceptions

CARPP Sample Program

CARPP is a C++ version of the CAR sample program. CARPP demonstrates how to create a Workplace Shell object using basic object-oriented programming techniques and the SOM, including:

- Initializing an object
- Adding settings pages to an object
- Saving and restoring the state of an object
- Modifying an object's context menus (adding and deleting menu items)
- Querying object class data
- Processing context menu items
- Implementing settings page dialog processing
- Handling exceptions

TEXTFLDR Sample Program

TEXTFLDR allows only plain text objects to be placed into the folder. Objects that are not plain text, that is, control- or extended-ASCII characters, are rejected. Objects are considered to be plain text if the "Associated type" field of the Settings notebook is set to plain text. If there is no associated type set, the first 500 bytes are placed into the folder.

WPSTUTOR Sample Program

WPSTUTOR displays the order in which object methods are invoked by the Workplace Shell. When a sample object's class method is executed, the method's name and a description of its function are displayed in a PM window. The sample saves the object's title and its icon title backwards.

This object class subclasses the WPDataFile class. Most Workplace Shell instance and class methods are overridden so that information about the methods can be displayed to the user.

This sample is designed to be more of a tutorial for Workplace Shell programming than a programming example.

Note: The SHOWDESC.EXE file should be renamed to SHOWDESC.BAK (or something else) to turn off the sample. Rename the file back

to SHOWDESC.EXE to turn the sample back on.

WSFILE Sample Program

WSFILE displays, creates, and installs Workplace objects of two classes: WSFILE and WSFOLDER. The WSFILE class is a subclass of the WPDataFile class, and the WSFOLDER class is a subclass of the WPFolder class.

Tools

This section provides a brief description of each tool with just enough detail to get you started at the OS/2 command line.

The tools are categorized as follows:

- Development Tools
- Multimedia
- Presentation Manager (PM)
- SOM
- TCP/IP
- Workplace Shell
- XPG4

For complete information about the tools described here, please refer to the online *Tools Reference* .

Note: The Wave Doctor, previously available with the OS/2 Multimedia Toolkit product, is not included in the OS/2 Warp Toolkit.

Development Tools

The Developer tools help you develop OS/2 programs. It is not likely that you will use the tools described in this section as much as the compiler and linker. Nonetheless, these tools are handy to have. Perhaps you will use only a few of them in your programming career, but it is important to know where you can find information about them, in the event you will ever need them. The tools included in this section are:

- ALP - Assembly Language Processor
- EXEHDR - Executable-Header Files
- FWDSTAMP - Forwarders
- IMPLIB - Import a Library
- KwikINF - Quick Information
- LINK386
- MARKEXE
- MASM2ALP - MASM Compatible Command Line Interface to ALP
- MKMSGF - Make Message File
- MKTMPF - Make Template File
- MSGBIND - Message Binding
- NMAKE
- PACK and UNPACK
- PACK2 and UNPACK2
- TRCUST - Trace Customizer

ALP

The Assembly Language Processor (ALP) is a macro assembler that runs under the 32-bit OS/2 operating system. ALP is designed as a functional replacement for the Microsoft Macro Assembler (MASM). It accepts the full syntax of the MASM 5.10 assembly language, which is a subset of the MASM 6.00 high-level directive language. It also understands the complete Intel 80X86 instruction set, including the MMX extensions. ALP creates standard Object Module Format (.OMF) files that can be linked to produce DOS or OS/2 executables and can generate line number debug information compatible with IBM's Presentation Manager Debugger. In addition, this tool offers a rich set of command-line options, as well as a comprehensive listing file with user-tailored formatting, allowing a visual perspective not possible with other assemblers.

ALP translates assembly language source files (typically having a file-name extension of .ASM) into object (.OBJ) files. The LINK386 utility can then be used to combine multiple object files into a single executable file, dynamic link library, or device driver.

While ALP is designed as a functional replacement for the Microsoft MASM assembler in terms of source code compatibility, it does not use the MASM command-line syntax. ALP uses a free-form syntax, has a comprehensive set of options, and allows assembly of multiple input files with a single command-line invocation.

For each corresponding input source file, ALP can produce the following types of output files:

- Object (.OBJ) files, which contain program data and executable code.
- Listing (.LST) files, which document the results of the assembly.
- Message (.MSG) files, which can contain all error, warning, and informational messages produced during the assembly.

See the *ALP Programming Guide and Reference* for more information on ALP. Online help is also available from the command line for all ALP options.

EXEHDR

EXEHDR provides a listing of the contents of the header of an executable file. It also provides a listing of the attributes of all segments in the file.

Uses of EXEHDR include:

- Determining whether a file is an application or a dynamic link library
- Modifying and viewing the attributes set by the module definition file
- Viewing the number and size of code and data segments.

FWDSTAMP

FWDSTAMP adds entry points, called *forwarders*, to a dynamic link library (.DLL) file. Forwarders point to API functions or other exported code or data. They contain an import reference so that the final target address of the forwarded entry is contained in a different module. A forwarder might be called an *imported export*.

When a file has a fix-up to a forwarded entry point, the loader resolves that fix-up to the address of the entry point that the forwarder imports by traversing the chain of forwarders until the end of the chain (a non-forwarded export) is reached. All forwarders are implicitly exported.

The imported entry point that a forwarder refers to may itself be another forwarder. The loader will process a chain of forwarders until a non-forwarder entry point is encountered.

There is no *run-time* cost to forwarders; however, there is a slight *load-time* cost as the loader resolves forwarder chains with their final addresses.

Using FWDSTAMP

You use forwarders to combine several DLLs into one without having to relink old applications. For example, if MOUCALLS and VIOCALLS were combined into a single DLL called NEWLIB.DLL, then MOUCALLS and VIOCALLS could be replaced with special DLLs containing

forwarders to NEWLIB.DLL.

IMPLIB

IMPLIB is used to generate an *import library* (.LIB) file. IMPLIB takes a module definition file (.DEF) as input. For each export definition in the .DEF file, IMPLIB generates a corresponding import definition.

The .LIB file generated by IMPLIB is used as input to LINK386, which creates an executable (.EXE) file. The .LIB file provides LINK386 with information about imported dynamic link functions.

Creating an IMPLIB

Import libraries are created by IMPLIB and are used to link DLLs with applications.

Import libraries are somewhat similar to standard libraries:

- You specify import libraries and standard libraries in the same command-line field of LINK386.
- Both types of libraries resolve external references at link time.

However, import libraries differ from standard libraries in that they contain no executable code. Rather, they identify the DLLs where the executable code can be found at run time.

Creating import libraries is an extra step. Nevertheless, import libraries are recommended for use with all DLLs for two reasons:

- IMPLIB automates much of the program creation process for you. To use IMPLIB, you need to supply the .DEF file you already created for the dynamic link library. Without an import library, you must create a second .DEF file that explicitly defines all needed functions in the dynamic link library.
- Import libraries make it easier for one person to write a library and another to write the application. Much of the linking process (linking the .DLL file and creating the import library) can be done by the author of the dynamic link library. The import library and associated .DLL file can then be given as a unit to the person linking the application--that person need not worry about creating a .DEF file.

KwikINF

KwikINF provides you with a quick and convenient method of accessing information in online documents stored in the OS/2 BOOKSHELF from anywhere on the Desktop, with the exception of DOS or WIN-OS/2 sessions. When KwikINF is started, you can search for a text string of your choice directly or through a pop-up window by pressing a user-selectable hot key. Until you configure KwikINF, your KwikINF hot key is Alt+Q.

How you initiate a search for information in online documents depends on where you are on the Desktop when you press the KwikINF hot key:

- From an OS/2 full-screen session, a PM VIO or AVIO window, or PM MLE:

  Position the cursor on the string you want to search for and press the KwikINF hot key. KwikINF retrieves the word at the cursor. If you have configured KwikINF to display the KwikINF window when the KwikINF hot key is pressed, KwikINF will automatically place the retrieved word in the **Search string** entry field of the KwikINF window. When you select **Search** or press Enter, KwikINF displays the information. If you have configured KwikINF to bypass the KwikINF window when the KwikINF hot key is pressed, KwikINF automatically displays the information.

- From a graphic-text PM window:

  Press the KwikINF hot key, and then type the string you want to search for in the **Search string** entry field of the KwikINF window. The KwikINF text-retrieval feature is not available from graphic-text PM windows.

LINK386

LINK386 is used to combine object files and standard library files into a single executable file. LINK386 also generates DLLs and device drivers.

LINK386 uses the following files as input:

- One or more object files that are linked with any optional library files to create the executable file. Object files usually have a .OBJ extension.
- One or more library files. The library files contain object modules that are linked with the object files to create the executable file. Library files usually have a .LIB extension.
- A module definition file. The module definition file provides information to LINK386 about the executable file or dynamic link library file it is creating. The module definition file usually has a .DEF extension.

LINK386 produces three types of output files:

- An executable file. This executable file runs under OS/2 whenever you specify a module definition file that has a NAME statement. The executable file usually has an .EXE extension.
- A dynamic link library file. A dynamic link library is produced whenever you specify a module definition file that has a LIBRARY statement. A dynamic link library file usually has a .DLL extension.
- A device driver file. A virtual or physical device driver is produced whenever you specify a module definition file that has the VIRTUAL DEVICE or PHYSICAL DEVICE statements. A device driver file usually has a .SYS extension.

Note:

Three additional options are available with the OS/2 Warp Toolkit:

/E[XEPACK:{1|2}]	Causes pages of code and data in the file to be compressed.
/NOO[UTPUTONERROR]	Prevents the LINK386 from creating the executable file if an error is encountered.
/NOS[ECTORALIGNCODE]	Turns off the alignment feature provided through the LINK386 linker. LINK386 normally aligns pages of code on sector (512 byte) boundaries, reducing the time to load the pages when the application is executed. The /NOSECTORALIGNCODE option aligns pages of code based on the value used in the /ALIGN option.

Creating a Response File for LINK386

To operate LINK386 using a response file, you must first create a file that contains the responses you want LINK386 to process. You can give the file any name and create it with any text editor.

Type the following command at the command prompt:

```
LINK386 @filename[.ext]
```

The @ symbol tells LINK386 that *filename* is a response file. If the file is not in the working directory, you must specify the path. Begin using a response file at any point on the LINK386 command line or at any LINK386 prompt. The file should contain responses in the same order as the LINK386 prompts. Each response needs to be on a separate line. If you choose to place responses on the same line, separate them with commas.

If the file does not contain responses for all the prompts, LINK386 displays the appropriate prompt and waits for you to supply a response.

End the response file with a semicolon.

You can use special characters in the response file the same way you would use them in responses entered at the keyboard.

Example of a Response File for LINK386

The response file in the following example instructs LINK386 to generate an executable file called FUN.EXE, and four object modules: FUN, SUN, RUN, and GAMES.

```
fun+sun+run+games /map
fun.exe
funlist
;
```

If you specify the file name FUNLIST, LINK386 will generate a map file named FUNLIST.MAP. Adding the /MAP option causes LINK386 to include the public symbols of the application in the map file.

OS2STUB.EXE

OS2STUB.EXE is included in the executable file created by LINK386 if the STUB statement is included in the module definition file. The stub is invoked whenever the file is executed under DOS. By default, LINK386 adds its own standard stub for this purpose.

MASM2ALP

MASM2ALP is a utility that accepts a Microsoft Macro Assembler (MASM) command-line syntax, translates the parameters into a syntax acceptable to the ALP assembler, and then invokes ALP.EXE to perform the assembly. It allows ALP to be used in place of MASM without requiring modifications to existing makefiles. See the *ALP Programming Guide and Reference* for more information.

MARKEXE

MARKEXE lets you view and set the type of application. The type of application identifies the OS/2 sessions in which a program can run. You can use MARKEXE in conjunction with programs that you have created using LINK386 or with programs created by some other means.

MKMSGF

MKMSGF converts a text message file to an output (binary) message file that DosGetMessage uses to display messages. Text messages in OS/2 full-screen applications do not need to be loaded into memory with the application; they can reside on disk until needed.

You can use the output message file by specifying a message file name and a message number in the DosGetMessage parameter list. The messages also can be bound to the executable file by MSGBIND.

Creating a MKMSGF

The input message file is a standard ASCII file that contains three types of lines:

- Comment
- Component identifier
- Component message

Comment lines are the first lines of a file and must begin with a semicolon. A component-identifier is a three-character name identifier (for example, "DOS") that precedes all MKMSGF message numbers. Component-message lines consist of a message header and an ASCII text message.

The following is an example of a text message source file.

```
;This is an example
;of a text message
;file
DOS
DOS0100E: File not found
DOS0101?:
DOS0102H: Usage: del [drive:][path] filename
DOS0103?:
DOS0104I: 1% files copied
DOS0105?:
DOS0106W: Warning! All data will be erased!
DOS0107?:
DOS0108?:
DOS0109P: Do you wish to apply these patches (Y or N)? %0
```

where:

DOS0100E - DOS0109P	Identifies message numbers in sequence. The first three characters indicate the component identifier; the four-digit number indicates the message number, which is followed by a letter (described below), and then a colon and blank space. If a message number is not used, type the number, end it with a question mark (?), and leave an empty entry.
E, H, I, P, W	Indicates the type of message. Categories include error (E), help (H), information (I), prompt (P), and warning (W).
%0	Displays a prompt for input from the user, after which a carriage return and line feed are inserted.

MKTMPF

MKTMPF creates template repository files from text input files. A template repository is a binary file used by the OS/2 Error Logging Facility to find error messages, causes, and actions in various message files. MKTMPF is located in the \TOOLKIT\BIN subdirectory.

MKTMPF is located in the \TOOLKIT\BIN subdirectory.

MSGBIND

When DosGetMessage is issued, it searches for the message in the message segment bound to the application's executable file, and then searches the application's message file on a hard disk. To ensure that a message is displayed quickly, you can bind it to the application's executable file by using the MSGBIND utility program. For each executable file, MSGBIND specifies which message files to scan; for each message file, it specifies which message to include in the executable file.

Note: The MSGBIND utility program supports the new compression format available with LINK386 (/EXEPACK:2) and RC (-x2).

The input file contains three types of lines:

- > Executable file
- < Message file
- Message numbers

An example of an input file follows:

```
>PROG1.EXE
<\MESSAGES\PRGMSG.MSG
PRG0100
PRG0101
PRG0102
<\MESSAGES\APP.MSG
APP0001
APP0002
APP0003
```

where:

>PROG1.EXE	Is the executable file to be modified
<	Defines the first message of a series to be bound, delimited either by the end of the series or a less-than symbol (<).
<\MESSAGES\PRGMSG.MSG and <\MESSAGES\APP.MSG	Names the files containing the binary versions of the messages (created by MKMSGF) and their identifying numbers: the three-character component identifier and the four-digit message number.

NMAKE

NMAKE carries out all tasks needed to create or update a program after one or more of the source files in the program have changed. NMAKE compares the modification dates for one set of files (the target files) with those of another set of files (the dependent files). NMAKE then carries out a given task only if a target file is out of date. NMAKE does not compile and link all files just because one file was updated. This can save time when creating programs that have many source files or that take several steps to complete.

Using NMAKE

To use NMAKE, create a description file (or makefile). A description file, in its simplest form, lists which files depend on others and which commands need to be executed if a file changes. You can create an NMAKE description file with any text editor that produces ASCII files.

A description file looks like this:

```
targets... : dependents...
     command                `     description block
        :
targets... : dependents...
     command
        :
```

A dependent relationship among files is defined in a *description block* . A description block indicates the relationship among various parts of the program. It contains commands to bring all components up-to-date. The description file can contain any number of description blocks.

Use NMAKE description files for creating backup files, configuring data files, and running programs when data files are modified.

PACK and UNPACK

PACK reduces the size of a file by compressing its data. You can use PACK for a single file or a group of files, thereby reducing the disk space required for your OS/2 application. UNPACK (provided with the OS/2 operating system) restores a compressed file to its original size and copies it to a specified directory.

Creating a List File

To use a list file with PACK, you must first create a file that contains the names of the files you want to compress. You can give the list file any name. Following is an example of specifying a list file at the command line:

```
PACK DEVICE.LST DEVICE.DRV /L
```

The /L indicates that DEVICE.LST is a list file. If the list file is not in the working directory, you must specify the drive and path. Global file-name characters are not permitted in the list-file name. DEVICE.DRV is the destination file for the end-to-end-compressed data. (End-to-end compressed data is the data from each of the files contained in the list file. This data is stored in a contiguous format in the destination file.)

The syntax used in the list file is similar to that used in the command line. The syntax for a single line in the list file follows:

```
sourcefile [/H:headerpath\|/H:headerfile|/H:headerpath\ headerfile]
           [/D:headerdate] [/T:headertime] [/C]
```

Remember, when using the list-file method (method 2), global file-name characters are not permitted in the source-file name. Notice also that "PACK" is excluded, and *packedfile* is not permitted in the list file, because they were specified on the command line. You can include comments or blank lines by entering a semicolon as the first character of the line. An example of a list file follows:

```
;This is a comment
C:\OS2\COMMAND.COM
CONFIG.SYS /H:CONFIG.BAK /C
\OS2\INSTALL\DDINSTAL.EXE /H:\OS2\DDINSTAL.TMP /D:10-15-91 /T:11.45
```

PACK2 and UNPACK2

The algorithm used to compress files has been substantially enhanced. The use of these utility programs is identical to the PACK and UNPACK utilities, which are documented in the *Tools Reference* .

Note: UNPACK2 is provided with the OS/2 operating system.

TRCUST, MAKETSF, and DEBDEL

The Trace Customizer utility program (TRCUST) converts a trace source file (TSF) to a trace format file (TFF) that defines the formatting for an event trace record. You can use TRCUST to combine multiple TFFs into a single TFF.

Two additional tools that are complementary to TRCUST are also included in the Toolkit:

- MAKETSF

  The purpose of MAKETSF is to extract dynamic trace definitions imbedded in C or ASM source file to which they relate. MAKETSF can also substitute line number information into those trace definitions that are specified by line number and source file reference.

- DEBDEL

  The purpose of DEBDEL is to remove debugging information from a load module after TRCUST has generated the trace definition file (TDF).

For more information on all of these tools, refer to the *OS/2 Debugging Handbook*.

Multimedia Tools

This section describes the Multimedia tools that help you develop multimedia programs. The following multimedia tools are available in this release:

- MIDICONV - MIDI Conversion Utility
- P2STRING

Note: The Wave Doctor, previously available with the OS/2 Multimedia Toolkit product, is not available with this version of the OS/2 Warp Toolkit.

MIDICONV

MIDICONV is a conversion utility program that lets you convert a MIDI format 1 file to a MIDI format 0 file through the use of an installable I/O procedure.

Note: A MIDI format 0 file has only one track of music; a MIDI format 1 file has multiple tracks of music.

P2STRING

P2STRING is used to test OS/2 multimedia subsystems through the media control interface string commands in the OS/2 multimedia environment. This tool processes script files (containing string commands and tool directives) to test the behavior of subsystems in OS/2 multimedia. P2STRING extracts the strings from the script files and processes the commands through the mciSendString function. Messages and error conditions of the processes included in the scripts are logged to an output file and displayed in windows.

P2STRING provides subsystem developers with an effective testing tool because it alleviates the need for extensive test code to be written. Developers can write script files to test all relevant scenarios and it also aids in debugging using log files.

Note: Refer to the README.P2S file in the P2STRING subdirectory for information about how to use P2STRING and how to create script files that it can process. The *OS/2 Multimedia Subsystem Programming Guide* contains additional information describing P2STRING.

Presentation Manager (PM) Tools

This section describes the Presentation Manager tools that help you develop Presentation Manager programs:

- DLGEDIT - Dialog Editor
- FONTEDIT - Font Editor
- ICONEDIT - Icon Editor
- IPFC - Information Presentation Facility Compiler

- RC - Resource Compiler

DLGEDIT

DLGEDIT (Dialog Editor) is used to create and modify dialog boxes and specify the controls and text within dialog boxes. As you create a dialog box and add controls, the dialog editor draws it to the screen. You can resize and reposition the dialog box, and then test its controls before you incorporate it in your application.

Although the dialog editor draws box outlines and controls to the screen so that you can view it from a user's perspective, the dialog editor does not save it as a graphic. Instead, the dialog editor stores a description of the dialog box and its controls in a text file that has a file-name extension of .DLG. It also creates a compiled form of the .DLG file into a resource file that has an extension of .RES. The dialog-box and resource files can each contain descriptions of more than one dialog box. The resource file can contain other application resources, such as icons, bitmaps, and string tables. It is attached to the application's executable (.EXE) file after the compile and link processes.

This version of the Dialog Editor is enhanced for use with Pen for OS/2. With this enhanced version, handwriting and sketch controls are available.

FONTEDIT

FONTEDIT (Font Editor) is used to design and save fonts for use in applications. A font is a set of alphanumeric characters, punctuation marks, and other symbols that share a common typeface design and line weight.

When the font editor creates a font file, it supplies an .FNT file-name extension. The font file contains a header, which describes the font in general terms, and a section that contains bitmaps of the characters themselves.

Font Resource Files

All resources, except fonts, can be bound to the application's executable file or compiled into a dynamic link library (DLL). Fonts must be put in a separate .DLL using the RC. You then link the file at run time and load the resources into your application by using DosLoadModule or GpiLoadFonts. A .DLL containing font resources must have a file-name extension of .FON. The .FON file can be installed on the system.

ICONEDIT

ICONEDIT (Icon Editor) is used to create icons, pointers, and bitmaps. In the PM, an icon is a graphic symbol that identifies a data object, a system action, or a minimized application. A pointer is a small shape on the screen that reflects the movement of the mouse. Pointers have a *hot spot* that identifies their exact location on the screen.

Icons, pointers, and bitmaps produced by the Icon Editor are graphic symbols comprised of pels in any of the following display states:

- Black
- White
- Color
- Screen (background color)
- Inverse screen (inverse of background color).

Mini-icons (also called small icons) are icons that are one-half the size of normal icons (also called large icons). Normal icons are 32x32 or 40x40 pels (depending on the monitor resolution). Mini-icons are 16x16 or 20x20 pels. They are used in areas where a normal icon is too large, for example, in toolbars.

If you previously had OS/2 installed on your system, and if you did not update your icon profile when you installed this OS/2 Warp Toolkit,

you need to run the Icon Editor once with its profile update options in order to use the mini-icon forms. To do this, enter:

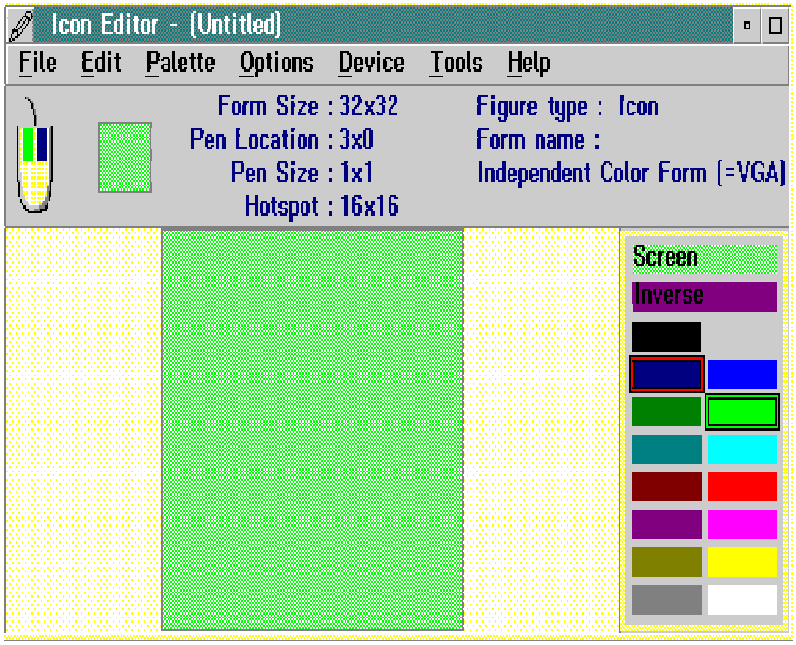
```
x:\TOOLKIT\BIN\ICONEDIT /t /i
```

where *x* is the drive location for the installed OS/2 Warp Toolkit.

After the initial run of Icon Editor, the new mini-icons will be part of your profile and you will not need to use options /t /i again.

Starting ICONEDIT

To start the Icon Editor, select the PM Development Tools folder, and then select the Icon Editor object. The following window is displayed:



Notice the information area at the top of the Icon Editor window; the items displayed from left to right include:

- A two-button mouse, showing the color currently selected for each button
- An actual-size image of the current figure that you are editing
- A status area that provides:

    - Size (in pels using X- and Y-coordinates)
    - Pen location
    - Pen size (from 1-by-1 to 9-by-9)
    - Hot spot (for icons and pointers, but not bitmaps)
    - Figure type (icon, pointer, or bitmap)
    - Form name

The palette window, in the lower-right corner, displays the colors that are available for use during editing. The colors currently selected are marked with frames.

The editing window is the largest part of your working area. Use the mouse to paint the enlarged representation with the selected color.

The menu-bar choices provide access to the many functions of the Icon Editor. These choices enable you to:

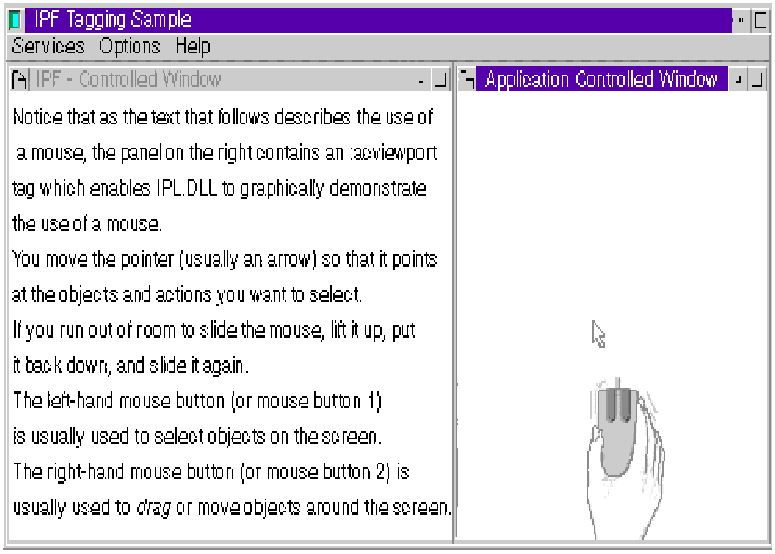
- Create a new icon, pointer, or bitmap
- Edit an existing icon, pointer, or bitmap
- Test the new icon or pointer
- Superimpose a grid over the editing window (for drawing a symmetrical figure)
- Restrict a drawing to straight vertical or horizontal lines

- Make transparent pels (for icons or pointers) visible
- Change the shape and size of the pen
- Select system preferences (to set prompts or suppress warnings)
- Define hot spots (where the mouse pointer is directed).

IPFC

Information Presentation Facility (IPF) is a set of tools used to create an online help facility for an application. IPF is also used to create online information that can be viewed independent of an application. It is a tool for both the information author and the application programmer.

As an author of online information, you can define the windows in which information is displayed. For example, a window can be split so that scrollable text can be displayed beside a stationary illustration that the text describes. The following figure shows an IPF application-control-window.



Developing Online Information

IPF makes it possible for you to design your information in two types of formats:

- An online document format for reference books
- A help-window format for context-sensitive help for your application programs

To produce either format, you must create a text file using a text editor and two IPF elements:

- IPF tagging language, which consists of instructions for formatting and displaying your document on the screen.
- Information Presentation Facility Compiler (IPFC), which interprets the tags and converts the source file into an IPF format.

Compiling Help Files

To compile a source file that is intended as a help-text window, use the IPFC command without the /INF option. For example:

```
IPFC myhelp.hlp
```

Compiling with International Language Considerations

The following parameters provide international language support:
 /COUNTRY = *nnn* (where *nnn* is the 3-digit country code)
 /CODEPAGE = *nnnn* (where *nnnn* is the 3 or 4-digit code page)
 /LANGUAGE = *xxx* (where *xxx* is a 3-letter identifier that
 indicates an international language file is to be used)

An example of the command-line syntax follows:

```
IPFC myfile.txt /INF /COUNTRY=033 /CODEPAGE=437 /LANGUAGE=FRA
```

You can now add additional languages to IPFC by providing an IPF*xxx*.NLS file where *xxx* matches the name of the language supplied with the /LANGUAGE or -L switch.

You can also add additional code pages by providing an APSY*xxxx*.APS file where *xxxx* matches the number of the code page. Four-digit code pages are supported. Three-digit code pages have a leading 0 in their APSY*xxxx*.APS file name, but the leading 0 does not have to be passed with the /CODEPAGE or -C switch. The underlying operating system must support the supplied code page.

Note: When adding new language or code page files, be sure they are located in the same subdirectory where your IPFC program files are located.

The bidirectional version of IPFC.EXE (IPFCBIDI.EXE) is not included as a separate program with this version of the OS/2 Warp Toolkit. However, the bidirectional support in IPFCBIDI has been integrated into the American version of the program, IPFC.EXE. It will be necessary for you to modify any makefiles that reference IPFCBIDI.EXE to reference IPFC.EXE instead.

IPFC Environment Variables

There are four environment variables that can be used to specify the location of source files:

IPFC	Specifies the location of the .IPF support files (such as APSYMBOL.APS and IPF*.NLS).
IPFCIMBED	Searches for files imbedded with the .im macro.
IPFCARTWORK	Specifies the location of artwork files and artlink files.
TMP	Indicates where the compiler should store the temporary files it creates during the compilation.

Viewing an Online Document

If you want to see your formatted online document, you can use the VIEW command to display it.

An online document has an extension of .INF. It can be viewed by entering its name as a parameter to the VIEW command; for example:

```
VIEW myfile
```

You do not need to include the .INF file extension.

Note: You cannot use VIEW to display help-text windows for application programs. However, for test viewing purposes, you can compile the help text as .INF files and use VIEW to look at the information.

RC

RC (Resource Compiler) is a tool that lets you add application resources, such as message strings, pointers, menus, and dialog boxes, to your application's executable file. The primary purpose of RC is to prepare data for applications that use functions such as WinLoadString, WinLoadPointer, WinLoadMenu, and WinLoadDlg. These functions load resources from the application's executable file or another specified executable file. The application then can use the loaded resources as needed.

RC and the resource functions let you define and modify application resources without recompiling the application itself. RC can modify the resources in an executable file at any time without affecting the rest of the file. You can create custom applications from a single executable file by using RC to add the custom resources you need to each application. RC is especially important for international language support because it lets you define all language-dependent data, such as message strings, as resources. Preparing the application for a new language is simply a matter of adding new resources to the existing executable file.

Creating an RC File

All resources are defined in a resource script file. You use a text editor to create a resource script file that has an .RC extension. Resources are defined either explicitly in statements in the resource script file, or in other files (such as output files from the resource editors). The .RC file is the input file to the RC; the output has an .RES extension. The .RC file can contain statements that define resources and that include resources from other files. Text-based resources such as menus, accelerator keys, and text strings are defined in the .RC file.
Non-text-based resources are specified in the .RC file as file names of the external files where these resources reside. Such resources include icons, pointers, and bitmaps. The syntax for including external files in a resource script varies according to the nature of the resources defined or contained in the files. Fonts have a resource file to themselves.

Make sure that none of the include files in your resource script file contain an end-of-file character. When the RC sees an end-of-file character, it assumes it to be the end of all input.

For an example of a resource script file, see the sample program TEMPLATE.

SOM Tools

SOM tools help you develop SOM programs. This section describes the SOM tools:

- CTOI
- IRDUMP - Information Repository Dump
- PDL - Public Definition Language
- PREGIMPL - PM version of REGIMPL
- REGIMPL - Implementation Registration
- SC - SOM Compiler
- SOMCORBA - SOM CORBA
- SOMDCHK - SOM Check
- SOMDD - SOM DSOM Daemon
- SOMDSRV - SOM Server
- SOMENV - SOM Environment
- SOMSTARS
- SOMXH - SOM .XH Header Files
- WPIDL2XH - Workplace Shell .IDL To .XH Files

CTOI

CTOI automates the conversion process from the SOM 1.0 format (.CSC files) to SOM 2.0 format (.IDL files). An example of the syntax follows:

```
CTOI f1
```

where *f1* is the name of the .CSC file to be converted.

You might require some modifications to your environment and your existing code in order to use this tool successfully. For more information, see the article titled *Using the SOM CTOI Tool* in the online version of the Developer Connection News, Volume 5. This newsletter can be found by opening the following folders, starting with The Developer Connection folder:

```
The Developer Connection Browser
  The Developer Connection for OS/2 References
    Newsletters
      Developer Connection News
```

IRDUMP

IRDUMP (Information Repository Dump) verifies that a class exists in the Information Repository. An example of the syntax follows:

```
IRDUMP [-o] [-w] [-?] [object]
```

where:

-o	Includes file offset information.
-w	Follows dump with "within" operation.
-?	Shows this usage information.
object	Is a simple or fully qualified name of an object in the Interface Repository.

The default action is to dump all objects.

PDL

PDL (Public Definition Language) is a separate program that performs the same function as the Public Definition Language (PDL) emitter used with the SOM Compiler. That emitter generates a copy of an .IDL file that has had the portions designated as private removed. The file generated is the same as the .IDL file from which it is produced, except that it removes all items within the .IDL file that are marked as "private." An item is marked as private by surrounding it with *#ifdef __PRIVATE__* and *#endif* directives. Thus, the PDL emitter can be used to generate a "public" version of an .IDL file. (Generally, client programs will need only the "public" methods and attributes.)

The PDL program can be invoked independently of the SOM Compiler. In addition, the PDL program can remove any kind of items in the .IDL file that are preceded by a user-specified *#ifdef* directive and followed by an *#endif* directive.

The PDL program is invoked as follows:

```
PDL [-c] [-d] [-f] [-h] [-s] [-/] files
```

where:

-c *cmd*	Specifies that, for each .IDL file, the PDL program is to run the specified system command. This command can contain a single occurrence of the string *%s* , which will be replaced with the source file name before the command is executed. For example, the option *-c sc -sh %s* has the same effect as issuing the SC command with the *-sh* option.

-d *dir*	Specifies a directory in which the output files are to be placed. (The output files are given the same name as the input files). If no directory is specified, the output files are named *<fileStem>.PDL* (where *fileStem* is the file stem of the input file) and are placed in the current working directory.
-h	Requests this description of the PDL command syntax and options.
-f	Specifies that output files are to replace existing files with the same name, even if the existing files are read-only. By default, files are replaced only if they have write access.
-s *smemit*	Specifies the SMEMIT variable with which the PDL program is to invoke the SOM Compiler.
-/ <string>	Specifies the *#ifdef* pattern for which the PDL program will strip out .IDL statements. The default value is *#ifdef __PRIVATE__* .
files	Specifies one or more .IDL files whose specified *#ifdef* sections are to be removed. File names must be completely specified (with the .IDL extension). If no *#ifdef* directive is specified (by including a *-/<string>* option), the *#ifdef __PRIVATE__* sections will be removed by default.

Selected options can be specified individually, as a string of option characters, or as a combination of both. Any option that takes an argument either must be specified individually or must appear as the final option in a string of option characters.

PREGIMPL

PREGIMPL is a Presentation Manager version of the REGIMPL tool.

REGIMPL

Before an implementation (a server program and class libraries) can be used by client applications, it must be registered with DSOM by running the Implementation Registration utility program, REGIMPL. This facility is available primarily for use in command (.CMD) files.

During execution of REGIMPL, DSOM updates its database to include the new server implementation and the associated classes. This enables DSOM to find and, if necessary, to activate the server so that clients can invoke methods on it.

Below are some examples on how you can use REGIMPL.

- To enter interactive mode:

```
REGIMPL
```

- To add implementation:

```
REGIMPL -A -i <str> [-p <str>] [-v <str>] [-f <str>] [-b <str>]
                    [-h <str>] [-m {on|off}] [-z <str>]
```

- To update implementation:

```
REGIMPL -U -i <str> [-p <str>] [-v <str>] [-f <str>] [-b <str>]
                    [-h <str>] [-m {on|off}]
```

- To delete implementation:

```
REGIMPL -D -i <str> [-i ...]
```

- To list implementations:

```
REGIMPL -L [-i <str> [-i ...]]
```

- To list aliases:

  ```
  REGIMPL -S
  ```

- To add classes:

  ```
  REGIMPL -a -c <str> [-c ...] -i <str> [-i ...]
  ```

- To delete classes:

  ```
  REGIMPL -d -c <str> [-c ...] [-i <str> [-i ...]]
  ```

- To list classes associated with implementations:

  ```
  REGIMPL -l [-i <str> [-i ...]]
  ```

  where:

-i <str>	Is the implementation alias name.
-p <str>	Is the server program name. The default value is SOMDSVR.EXE.
-v <str>	Is the server-class name. The default value is SOMDServer.
-f <str>	Is the reference data file name. Use NULL to delete.
-b <str>	Is the reference data backup file name. Use NULL to delete.
-h <str>	Is the host machine name. The default value is localhost.
-m {on|off}	Enables multi-threaded server.
-z <str>	Is the implementation ID.
-c <str>	Is the class name.

SC

The OS/2 operating system provides a programming interface that enables applications to implement Desktop objects. This programming interface enables you to create Desktop objects that conform to the CUA architecture using a basic object-oriented programming interface. The interface is implemented using the IBM SC (SOM Compiler).

The SOM Compiler (SC) helps implementers build classes in which interface and implementation are decoupled. The SOM Compiler reads the IDL definition of a class interface and generates:

- An implementation skeleton for the class
- Bindings for implementors
- Bindings for client programs

Bindings are language-specific macros and procedures that make implementing and using SOM classes more convenient. These bindings offer a convenient interface to SOM that is tailored to a particular programming language. For instance, C programmers can invoke methods in the same way they make ordinary procedure calls. The C++ bindings "wrap" SOM objects as C++ objects, so that C++ programmers can invoke methods on SOM objects in the same way they invoke methods on C++ objects. In addition, SOM objects receive full C++ typechecking, just as C++ objects do. Currently, the SOM Compiler can generate both C and C++ language bindings for a class.

SMEMIT Environment Variable

SMEMIT is used to indicate which emitter programs should be executed. Like the SMINCLUDE environment variable it can consist of a list of items separated by semicolons. Each item designates a particular emitter by the name of the file extension the emitter produces. The syntax is as follows:

```
SMEMIT=[h;ih;c;xh;xih;xc;def;ir;pdl]
```

The default values are *h* and *ih*. For example:

```
SET SMEMIT=H;IH;DEF;
```

indicates that EMITH.EXE, EMITIH.EXE, and EMITDEF.EXE programs should be executed to produce .H, .IH, and .DEF files, respectively. By default all emitted output files are placed in the same directory as the input file. If the SMEMIT environment variable is not defined, the SOM compiler will perform a syntax check of your class definition but no output will be produced.

Note: All command-line modifiers can be set in the environment by changing them to UPPERCASE and preappending SM to them.

SMINCLUDE Environment Variable

The SOM compiler uses an environment variable called SMINCLUDE to locate included class definitions. SMINCLUDE specifies where to search for .IDL and .EFW files. Because every SOM class will have an include file for its parent class definition, you must set SMINCLUDE before running the SOM compiler. Its form is similar to the OS/2 PATH or DPATH environment variables, in that it can consist of one or more directory names, separated by a semicolon. Directory names can be specified with absolute or relative path names. The syntax is as follows:

```
SMINCLUDE=<dir1>[;<dir2>]+
```

For example:

```
SET SMINCLUDE=.;C:\TOOLKIT\SOM\INCLUDE;C:\TOOLKIT\IDL;
```

Note: All command-line modifiers can be set in the environment by changing them to UPPERCASE and preappending SM to them.

SMKNOWNEXTS Environment Variable

SMKNOWNEXTS add headers to user-written emitters. The syntax is as follows:

```
SMKNOWNEXTS=ext[;ext]+
```

Note: All command-line modifiers can be set in the environment by changing them to UPPERCASE and preappending SM to them.

SMTMP Environment Variable

The SMTMP environment variable specifies the name of a directory that the SOM compiler uses to hold intermediate output files. The syntax is as follows:

```
SMTMP=<dir>
```

For example:

```
SET SMTMP=%TMP%
```

tells the SOM compiler to use the same directory for temporary files as given by the setting of the TMP environment variable. As a general rule, the directory indicated by SMTMP should never coincide with the directory used by the SOM compiler for its input or the emitted output files.

Note: All command-line modifiers can be set in the environment by changing them to UPPERCASE and preappending SM to them.

SOMIR Environment Variable

SOMIR provides a list of IRs to search for. The syntax is as follows:

```
SOMIR=<path>[;<path>]+
```

Note: All command-line modifiers can be set in the environment by changing them to UPPERCASE and preappending SM to them.

#pragma Directives

There are two pragma directives:

#pragma somemittypes on	Turns on the emission of global types.
#pragma somemittypes off	Turns off the emission of global types.
#pragma modifier <modifier stm>	Replaces a modifier statement.

Running SOM Emitters

You complete the SOM compilation process by running the emitters. You can control the output of the emitters from the command line by typing:

command [-o *filename*] [-a *name*[=*value*]]*

where:

command	Is one of the following: • EMITH • EMITIH • EMITC • EMITDEF
filename	Is an explicit name (including drive, path, and file-name extension) for the emitted output file. If this option is not specified, the output file is placed in the current directory, and the file-name extension defaults to a type appropriate to the selected emitter program.
-a *name* [=*value*]	Adds a global attribute.

SOMCORBA

SOMCORBA is a command script that you can use to convert to implicit pointers (like CORBA) for interface references.

SOMDCHK

SOMDCHK evaluates the environment to verify whether DSOM can operate correctly. The program generates messages that evaluate the DSOM environment. It determines whether the necessary SOM DLLs can be located, whether DSOM is enabled for workgroup (cross-machine) communication, whether Interface and Implementation Repositories can be located, and it displays the settings of important environment variables. In its "verbose" mode, SOMDCHK gives the default settings for DSOM environment variables and explains how DSOM uses them.

The program is invoked from the command line using the syntax given below. The optional verbose setting can be turned on by including the *-v* option with the following command:

```
SOMDCHK [-v]
```

SOMDD

SOMDD is the DSOM daemon. It must be started prior to running a DSOM application. The daemon can be started manually from the command line, or it could be started automatically from a start-up script run at boot time. It can be run in the background with the following command:

```
START SOMDD
```

The SOMDD program requires no parameters. An optional *-q* parameters can be used to set "quiet" mode, to suppress messages.

Note: The SOMDD.EXE program included with VisualAge C++ is incompatible with the version of SOM used with OS/2 Warp Toolkit and should not be used. You will need to update both your LIBPATH and PATH statements to ensure that the OS/2 Warp Toolkit path is ahead of the VisualAge C++ compiler.

SOMDSVR

SOMDSVR is the "generic" server program. It can be started either from the command line or automatically upon demand. When starting SOMDSVR from the command line, the server's implementation ID or alias must be supplied as an argument. The command syntax for starting a generic SOM server is:

```
SOMDSVR [impl_id | -a alias]
```

SOMENV

SOMENV is a command script to set the environment variables required for SOM programming. This command script requires that the SOMBASE environment variable be set.

SOMSTARS

SOMSTARS is a command script that you can use to convert to explicit pointers for interface references.

SOMXH

SOMXH is a command script that you can use to create the .XH header files from the .IDL files located in the \TOOLKIT\SOM\INCLUDE subdirectory.

WPIDL2XH

WPIDL2XH.CMD is a command file to emit .XH header files from Workplace Shell .IDL files.

To re-generate the .XH headers for C++, invoke this command file on each Workplace Shell class .IDL file from the Toolkit. It is only necessary to do this when you upgrade to a new level of SOM. Invoking the SOMXH command script will create .XH files for you, as the Workplace Shell classes currently only maintain passthru sections for .H files for C. The syntax is as follows:

```
WPIDL2XH <inputfile>
```

TCP/IP Tools

The following TCP/IP tools are available in this release of the OS/2 Warp Toolkit:

- RPCGEN
- WSFORMAT

RPCGEN

Use the RPCGEN command to generate C code to implement an RPC protocol. The input to RPCGEN is a language similar to C, known as RPC language.

You normally use RPCGEN infile to generate the following four output files. For example, if the infile is named PROTO.X, RPCGEN generates:

- A header file called PROTO.H
- XDR routines called PROTOX.C

- Server-side stubs called PROTOS.C
- Client-side stubs called PROTOC.C

RPCGEN is located in the \TOOLKIT\BIN subdirectory.

WSFORMAT

WSFORMAT is the WinSock 1.1 for OS/2 trace formatter. The WSFORMAT command accepts binary trace input from a serial port, file, or an OS/2 pipe, and converts the input into a readable format that can be displayed on the screen or written to a file, serial port, or OS/2 pipe.

VoiceType Tools

The Grammar Compiler takes a plain-text grammar file (BNF) and compiles it into a finite state grammar file (FSG). The FSG file is what the speech engine uses at run time to determine which words and phrases are currently available for a user to say to your application.

Documentation on the Grammar compiler can be found in the *Grammar Guide* . The VoiceType Developer's Toolkit also includes the following utilities to assist you in evaluating grammars:

- FSGENUM.EXE - Enumerates the strings produced by the grammar.
- FSGTEST.EXE - Determines whether input strings are accepted by the grammar.
- FSGPRINT.EXE - Displays the graph that defines the grammar.
- TIDYGRAM.EXE - Used to combine the Finite State Grammer (FSG) and What Can I Say (WCS) files, allowing grammars to be displayed in the What Can I Say window.

The Grammar compiler and utilities are located in the TOOLKIT\SPEECH\BIN subdirectory.

Workplace Shell Tools

This section describes the Workplace Shell tools that help you develop Workplace Shell programs. There are two Workplace Shell tools in this release :

- OBJUTIL - Object Utility/2
- WPCLSLST - Workplace Class List

OBJUTIL

OBJUTIL (Object Utility/2) is a new Workplace Shell tool that provides a facility for registering classes, and creating and modifying instances of classes.

WPCLSLST

WPCLSLST (Workplace Class List) creates a workplace object class and an instance of a workplace object class. Workplace objects are

constructed using the SOM protocol and are instances of one of the following workplace object classes:

Predefined	These classes are defined by the system. Examples of predefined workplace object classes are WPObject, WPFileSystem, and WPAbstract.
Subclass	These classes are derived from existing predefined workplace object classes. They add or remove function; however, they retain the basic behavior of that class.
Replaced	These classes replace the class being subclassed. They notify the behavior of an instance of a predefined workplace object class without the instance being aware of the new class.

XPG4 Tools

The OS/2 Warp Toolkit supports the development of fully internationalized, single source, single object (SSSO) applications that can be translated into other languages. This support is now enhanced by the addition of two new internationalization (I18N) models: an implementation of industry standard XPG4 I18N functions and tools and an implementation of the IBM Universal Language Support (ULS) functions.

Internationalization of programs requires that messages be displayed in the language of the user. This requirement is satisfied by producing versions of the messages that are translated into all supported languages. The OS/2 C library messaging support is based on the messaging support described in the X/Open XPG4 specification. It consists of functions for extracting messages from message catalog files (catopen, catgets, catclose) and the following utilities for producing message catalog files:

- GENCAT - Generate Message Catalog Utility
- MKCATDEF - Preprocess Message Source File Utility

The functions for extracting message from message catalog files are found in the *C Library Reference* .

GENCAT

The GENCAT utility processes a message source file and produces a catalog file usable by the C library messaging functions.

GENCAT creates the message catalog (usually *.cat) from a message source file (usually *.msg) or standard input. You can specify any number of message source files. A message source file is a text file that contains messages consisting of a message number followed by the message text. See the Preprocess Message Source File Utility (MKCATDEF) for using symbolic message identifiers in your message source file.

MKCATDEF

The Generate Message Catalog Utility (GENCAT) does not accept symbolic message identifiers; you must use the MKCATDEF utility if you want to use symbolic message identifiers in your message source file for use with GENCAT.

MKCATDEF processes a message source file containing symbolic identifiers and produces the following output:

- The SYMBOLNAME.H file, containing statements that equate symbolic identifiers with the set numbers and message ID numbers assigned by MKCATDEF. You must include the SYMBOLNAME.H file in your application program to associate the symbolic names to the set and message numbers assigned by MKCATDEF.
- A new message source file containing message numbers instead of symbolic message identifiers. This output is suitable for input to GENCAT.

MKCATDEF sends the new message source file, with numbers instead of symbolic identifiers, to standard output. You can use MKCATDEF output as input to GENCAT in the following ways:

- Use MKCATDEF with a > (redirection symbol) to write the new message source to a file. Use this file as input to GENCAT.

- Pipe the MKCATDEF output file directly to GENCAT.

After running MKCATDEF, you can use symbolic names in an application to refer to messages.

Online Documentation

The Toolkit provides a variety of online documentation for your needs. The OS/2 Warp Online Technical Library provides both guidance and reference information. This information can help you develop applications for OS/2 Warp.

The following topics describe the different online documentation.

OS/2 Warp Online Technical Library

The following online books are located in the \TOOLKIT\BOOK subdirectory. You can also access them from the Desktop by opening the Toolkit folder and then the Toolkit Information folder. The titles in the list below are the same as the associated book icon names in the Toolkit Information folder.

- ALP Programming Guide and Reference
- Bidirectional Language Programming Guide
- C Library Reference
- Control Program Programming Guide and Reference
- DMI Programmer's Guide
- DPI Programmer's Guide
- GPI Guide and Reference
- IBM OS/2 Server Family Programming Reference
- IPF Programming Guide and Reference
- Linear Executable Module Format Reference
- Multimedia Application Programming Guide
- Multimedia Programming Reference
- Multimedia Subsystem Programming Guide
- Object Module Format Reference
- Object REXX Programming Guide for OS/2
- Open32 Programming Guide and Reference
- OpenGL Programming Reference
- OS/2 Debugging Handbook
- OS/2 Programming Guide and Reference Addendum for OS/2 Warp Server for e-business
- Presentation Manager Programming Guide and Reference
- Problem Determination Guide and Reference
- RXSTRING.DOC
- SOM Programming Guide
- SOM Programming Reference
- TCP/IP Programming Reference
- Tools Reference
- Unicode Programming Reference
- Windows Sockets
- Workplace Shell Programming Guide
- Workplace Shell Programming Reference

How to Use Online Documents

The online documents in the OS/2 Warp Toolkit were developed with IPF. IPF displays information through a familiar user interface and lets you do the following:

- View a table of contents from which you can quickly gain access to a category
- View the category and select related topics from a menu
- View multiple windows of related information for comparison values
- Search for a topic throughout the document
- Copy the contents of a topic to the system clipboard for editing with the OS/2 System Editor, the Enhanced Editor, or any other editor with this capability
- Copy the contents of a topic to a temporary file for editing with a text editor

When installed, the online documents are added to the Toolkit Information folder. To access the online documents, double-click on the folder, and then select the appropriate book. A window that has a table of contents (Contents window) is displayed.

When the Contents window is displayed, some categories have a plus sign (+) beside them. The plus sign indicates that additional topics are available. Using the left mouse button, click on the plus sign to expand the category.

For more information about using the OS/2 Warp Toolkit online documents, refer to the How to Use This Book section.

ALP Programming Guide and Reference

This book describes how to install and run the ALP assembler. It provides a complete description of the following:

- Installing ALP
- Using ALP
- Language Reference
- Processor Reference
- Assembler Messages
- Return Codes

Bidirectional Language Programming Guide

This book provides information about the interface (API) for the Arabic and Hebrew languages to help programmers create Arabic or Hebrew applications for this environment.

The Bidirectional support in PM is provided in the Arabic and Hebrew language versions of the OS/2 operating system. The support is active/configured when the COUNTRY selection during OS/2 installation is Arabic or Hebrew in these versions.

C Library Reference

This book describes the C library functions provided as part of OS/2 Warp.

Control Program Programming Guide and Reference

This book provides the C-language syntax for each of the base operating-system application programming interfaces (APIs), including input and output parameters, data structures, data types, return codes, and example codes. Guidance information is also provided to assist you in developing applications using these items. API functions (indicated by the prefix "Dos") are presented by component, such as Error Management, Exception Management, and File System. The API functions, within each of the components to which they apply, are listed in alphabetic order. API functions also are available from a single alphabetic list.

DMI Programmer's Guide

This book is designed as a programmer's guide for the System Management Agent, which provides access to system components that have been defined according to the Desktop Management Interface (DMI) standard. These components can be hardware or software in the system that have been defined in the Management Information Format (MIF). Although the DMI itself is protocol-independent, the System Management Agent can manage any DMI-enabled components in the system and translate the MIF information into SNMP management information bases (MIBs).

DPI Programmer's Guide

This book is designed as a programmer's guide for the System Management Agent, which provides the Simple Network Management Protocol (SNMP) distributed protocol interface (DPI). The SNMP DPI permits users to dynamically add, delete, or replace management variables in the local Management Information Base (MIB) without requiring you to recompile the SNMP agent.

GPI Guide and Reference

This book describes the concepts associated with graphical output-- presentation spaces, device contexts, graphic primitives, fonts--and how to prepare graphical output for display and printing. It provides the C-language syntax for all the graphical programming interface (GPI), including input and output parameters, data structures, data types, return codes, and example codes. Guidance information is also provided to assist you in developing applications using these items. GPI functions (indicated by the prefix "Gpi") are listed in alphabetic order.

IPF Programming Guide and Reference

This book describes the concepts--help windows, hypertext linking, author- controlled viewports, dynamic data formatting--and the functions used for implementing help in OS/2 applications. It describes how to create online help and information. It also contains an alphabetic list of IPF tags, symbols, and control words. The IPFC error messages, window functions, dynamic data formatting functions, and help manager messages and functions are included.

Linear Executable Module Format Reference

This document describes the IBM OS/2 16/32-bit Linear eXecutable Module Format (LX). This is the load module format understood by the OS/2 32-bit system loader (for OS/2 version 2.0 and greater). LX load modules are created by Linear Executable linker utilities (such as IBM LINK386). A Linear Executable linker must be used in order to create 32-bit (flat-model) OS/2 programs; however, the LX format also allows for any combination of 16-bit and 32-bit code or data sections to exist within the same module.

IBM OS/2 Server Family Programming Reference

This reference provides a description of the application programming interfaces (APIs) provided by the IBM Operating System/2 for OS/2 Warp Servers, LAN Servers, and the Directory and Security Server (DSS). These functions provide the functional interface for server and client application development. The OS/2 operating system referred to in this book is OS/2 Versions 2.1 through OS/2 Warp 4 (with the applicable CSDs and ServicePaks applied).

This book also provides the OS/2 Warp LAN Requester API. These functions are 32-bit and are generally consistent with the Net32xxx and

Dos32xxx LAN functions that are included with LAN Server 4.0. DOS refers to DOS versions 3.3, 5.0, 6.x, or 7.0.

This book is written as a reference for the application or system programmer who is developing Directory and Security Server (DSS) software applications or OS/2 LAN Server software for a LAN Server, LAN Requester, OS/2 Warp Connect LAN Requester (client only), or DOS LAN Services (DLS) workstation. Information in this book applies to the following servers and clients:

- LAN Server 3.0
- LAN Server 4.0
- OS/2 Warp Server
- OS/2 Warp Connect
- Directory and Security Server

This book includes DLS programming information that is not supported by Directory and Security Server.

The reader should be familiar with programming the OS/2 base operating system or an equivalent multitasking operating system and should understand the C programming language.

Multimedia Application Programming Guide

This book describes the concepts associated with managing audio and video data and hardware using an extendable architecture that includes logical media devices (amplifier-mixer, waveform audio, MIDI sequencer, CD-audio, CD-XA, digital video, and videodisc) and I/O procedures for supporting various file formats.

Multimedia Programming Reference

This book describes the media control interface, PM graphic push buttons, secondary windows functions, multimedia I/O services, direct interface video extensions (DIVE), and subsystem services for synchronization and streaming.

Object Module Format Reference

This document describes the IBM 16/32-bit Object Module Format. Files of this format are produced by compilers and assemblers during the process of translating source code to object code. The resulting OMF object files are then combined and converted into executable program modules and/or dynamic link libraries by linker utilities such as LINK386.

Multimedia Subsystem Programming Guide

This book describes the subsystem components--media control driver, stream handler, and I/O procedure--that support a multimedia device.

Object REXX Programming Guide for OS/2

The *Object REXX Programming Guide for OS/2* describes the Object-based REstructured eXtended eXecutor, or Object REXX programming language. (When not comparing it to its traditional predecessor, it is just referred to as REXX.) REXX is an integral part of the OS/2 operating system.

This information is aimed at developers familiar with OS/2 who want to use REXX to do object-oriented programming--or some mix of traditional and object-oriented programming--with the shortest learning curve possible.

This information assumes you are already familiar with the techniques of traditional structured programming and uses them as a springboard for explaining REXX and, in particular, Object REXX. This "no frills" approach is designed to help experienced programmers get involved quickly with the REXX language, exploit its virtues, and become productive fast.

Open32 Programming Guide and Reference

The *Open32 Programming Guide and Reference* provides information on the following topics:

- What Open32 is and how it can help you to either:
  - Migrate Windows code to OS/2 code
  - Write common source code for OS/2 and Windows
- How to use the SMART tool to analyze Windows code and see how much effort is involved to migrate it to OS/2 code
- Differences in behavior between some Open32 functions and their Windows counterparts
- The functions supported by Open32

As an enhancement for Open32, the OS/2 Resource Compiler (RC) now supports string IDs for resources in addition to numbers. You can use either SMART or the Resource Compiler to convert resources. If you use the Resource Compiler, you do not need to include the hhh file with your code. OS/2 supports string IDs that contain quotation marks. See the discussion of the RESOURCE statement in the *Tools Reference* for more information.

OpenGL Programming Reference

This book provides an overview of OpenGL function and describes in detail the window and input interface that integrates OpenGL with the Presentation Manager. Application developers can use this interface to:

- Create and prepare an OpenGL context for rendering
- Swap a window's front and back buffers
- Select a color palette for a color index context
- Integrate GPI and OpenGL rendering and fonts

The interface provides much the same functionality as the OpenGL window interfaces for X Windows and Microsoft systems.

OS/2 Debugging Handbook

This book explains problem debugging in an OS/2 environment. The book is written for service personnel, system programmers, and software developers. It provides information about the internal workings of the OS/2 operating system and describes the use of system debugging tools.

OS/2 Programming Guide and Reference Addendum for OS/2 Warp Server for e-business

This book provides information about APIs that are new to OS/2 Warp Server for e-business. This information includes syntax, input and output parameters, data structures, data types, return codes, and sample code.

Presentation Manager Programming Guide and Reference

This book provides the C-language syntax for the base operating-system application programming interface (API), including input and output parameters, data structures, data types, return codes, and example codes. API function prefixes include Drg (dragdrop), Ddf (dynamic data format), Prf (profile), Spl (spooler), and Win (window). Also included are application hooks and PM messages.

Problem Determination Guide and Reference

This book provides programmers with guide and reference information for collecting and managing problem determination data.

This book first provides conceptual and introductory information about OS/2 Warp's First Failure Support Technology (FFST). This sophisticated and powerful tool requires thoughtful planning and code instrumentation. These topics present planning, setup, and examples that are used for collecting and managing problem determination data.

Separate sections are devoted to understanding the aids that are provided for tracking, collecting, storing, and formatting problem determination data, such as traces, dumps, and error logs.

After reading this book you should understand the benefits of FFST and know how to instrument your code. This understanding enables you to take advantage of the technology and tools that are associated with problem determination data. You should also know how to use the API, trace, error logging, and dump functions. These functions retrieve, format, and analyze problem determination data.

This book assumes that you have the OS/2 Toolkit installed and you are developing application programs.

RXSTRING.DOC

RXSTRING.DOC is an ASCII text file that documents the utility library that is included with the OS/2 Warp Toolkit. The utilities enable C programmers to effectively handle REXX string variables.

SOM Programmers Reference

This book is a complete reference for each of the classes and methods used for the object-oriented programming environment. Also included are System Object Model (SOM) C-language and C++ bindings, the Object Interface Definition Language syntax, and the SOM compiler command syntax.

SOM User's Guide

This book explains how programmers using C, C++, and other languages can:

- Implement class libraries that exploit the SOM library-packaging technology
- Develop client programs that use class libraries that were built using SOM
- Develop applications that use the frameworks supplied with the SOMobjects Developer Toolkit, class libraries that facilitate development of object-oriented applications.

TCP/IP Programming Reference

This book describes the routines for application programming in the TCP/IP for OS/2 Warp environment on a workstation.

Tools Reference

This book describes the tools that are included in the IBM Developer's Toolkit for OS/2 Warp.

Unicode Programming Reference

The Unicode Programming Reference documents the Universal Language Support (ULS) and Unicode conversion functions. OS/2 Warp Version 4 or later provides a dynamically linked C-language run-time library that supports multithreaded and Unicode-enabled programs. For information on this library, see the *C Library Reference* .

Windows Sockets

The Windows Sockets specification defines a network programming interface for Microsoft Windows, which is based on the "socket" paradigm popularized in the Berkeley Software Distribution (BSD) from the University of California at Berkeley. It encompasses both familiar Berkeley socket style routines and a set of Windows-specific extensions designed to help the programmer to take advantage of the message-driven nature of Windows.

The Windows Sockets Specification is intended to provide a single API to which application developers can program and multiple network software vendors can conform. Furthermore, in the context of a particular version of Microsoft Windows, it defines a binary interface (ABI) such that an application written to the Windows Sockets API can work with a conformant protocol implementation from any network software vendor. This specification thus defines the library calls and associated semantics to which an application developer can program and which a network software vendor can implement.

Network software which conforms to this Windows Sockets specification will be considered "Windows Sockets Compliant." Suppliers of interfaces that are "Windows Sockets Compliant" are referred to as "Windows Sockets Suppliers." To be Windows Sockets Compliant, a vendor must implement 100% of this Windows Sockets specification.

Applications that are capable of operating with any "Windows Sockets Compliant" protocol implementation are considered as having a "Windows Sockets Interface" and are referred to as "Windows Sockets Applications."

This version of the Windows Sockets specification defines and documents the use of the API in conjunction with the Internet Protocol Suite (IPS, generally referred to as TCP/IP). Specifically, all Windows Sockets implementations support both stream (TCP) and datagram (UDP) sockets.

While the use of this API with alternative protocol stacks is not precluded (and is expected to be the subject of future revisions of the specification), such usage is beyond the scope of this version of the specification.

Workplace Shell Programming Guide

This book describes the concepts associated with object-oriented programming for the OS/2 operating system--SOM, Workplace Shell classes and methods--and how to create object-oriented applications for the OS/2 Desktop.

Workplace Shell Programming Reference

This book provides the detailed descriptions of the Workplace Shell object-oriented programming interface.

VoiceType Developer's Toolkit Documentation

Online documentation for the VoiceType Developer's Toolkit is located in the \TOOLKIT\SPEECH\BOOK subdirectory. You can also access the information from the Desktop by opening the Toolkit folder, and then the VoiceType Developer's Toolkit folder.

Note: The VoiceType Developer's Toolkit online documentation is in HTML format. You need a Web browser to view it.

The following information is provided:

- Application Programming Interface (API) Reference
- Grammar Guide

API Reference

This reference provides a description of the Speech Manager (Sm) application programming interface. Two types of functions are covered--function calls to the speech engine and data access functions, which are functions that retrieve data from reply messages.

Grammar Guide

Grammars can be used by speech-enabled applications to specify what phrases to accept. Grammars can also be used to enhance Voice Manager navigation of applications that are not speech-enabled. The *Grammar Guide* describes the basic steps for writing an application grammar. It also contains information that explains how to use the grammar utilities that are provided with the VoiceType Developer's Toolkit.

Programming Considerations

Programming considerations apply for the following areas of the OS/2 Warp operating system:

- General Considerations
- Presentation Manager
- SOM
- IBM VisualAge C++ for OS/2 Compiler
- Workplace Shell

General Considerations

Included in this section are important items that can affect your future development efforts when using the IBM Developer's Toolkit for OS/2 Warp.

Compiling with IBM C/2

The IBM Developer's Toolkit for OS/2 Warp Version 4 and later versions are not intended to be used with the IBM C/2, Version 1.1 compiler. If you choose to use that compiler, you should use version 1.3 of the Toolkit.

The OS/2 Warp Toolkit is intended to be used with the IBM VisualAge C++ compiler.

Compiling with IBM VisualAge C++ for OS/2

The latest version of the C Set ++ compiler (now called VisualAge C++) has been released. VisualAge C++ includes its own version of the OS/2 Warp Toolkit, which includes updated sample makefiles. Changes to the sample makefiles relate to the new linker (ILINK) and to the new library (CPPOM30.LIB).

With this version of the OS/2 Warp Toolkit, the same sample makefile changes have been made so that you can use VisualAge C++.

You can compile the samples included with the OS/2 Warp Toolkit with C Set ++ (using LINK386) by changing any reference to CPPOM30.LIB to:

```
DDE4MBS.LIB
```

and changing any reference to ILINK to:

```
LINK386
```

Note: The */nofree* option must be removed from the LINK386 statement.

Installing IBM VisualAge C++ for OS/2

Installing your compiler first and the OS/2 Warp Toolkit last prevents unexpected results due to environment variable changes (automatic updates to CONFIG.SYS).

To avoid a downlevel SOM Runtime installed by the VisualAge C++ 3.0 compiler, follow these steps for installation:

1. Start the installation program for VisualAge C++ 3.0.
2. Deselect the OS/2 Warp Toolkit entries in the installation screen and proceed with the installation.
3. After completing the installation of VisualAge C++ but before restarting the system, edit CONFIG.SYS and modify the LIBPATH statement by moving x:\ IBMCPP\DLL after y:\OS2\DLL, where *x* is the drive where the VisualAge C++ compiler is installed and *y* is the drive where the OS/2 Warp operating system files are installed. This prevents the SOM Runtime installed by VisualAge C++ from overriding the SOM Runtime shipped with OS/2 Warp Version 4 or later.

4. Save the modified CONFIG.SYS and restart your system.

5. Install the OS/2 Toolkit and restart your system.

Known Limitations for the Container Part

The Container part has the following known limitations. Note that the behavior described below will be exhibited by all samples that subclass from the Container part.

- A SYS3175 error occurs when you select **Open As->Details** from the Container part's View menu. This error will close the document.
- A SYS3175 error occurs when saving a document containing a selected Container part inside a root Container part. It will occur if the selected part subclasses from the Container part. This error will not occur if the selected part, for example, is the Cookbook part.
- The Container part's **Show As** menu selection from the View menu does not work correctly. This selection will incorrectly change the view type of selected parts, when it should only change the activated part's view type. The correct way to change the view type of selected parts would be to select **Show selection as** from the Edit menu.
- The **Move** selection from the Edit menu is incorrectly enabled for an activated part. The **Move** selection, like all Edit menu selections, are for selected parts only.
- A part that subclasses from the Container part may not have its intrinsic content clipped correctly to the embedded parts. The Container part correctly handles the clipping of its embedded parts, but there is no way for a part that subclasses from the Container part to tell the Container part how to clip the embedded parts to its own content. Generally, the subclass part's intrinsic content will write over the embedded parts.

Known Limitations for DIVE

DIVE will not run in direct mode on some systems if the video card requires bank switching. To determine if your video card requires bank switching, select **Query Caps** from the Options menu of the DIVE sample. If the **Screen access requires bank switch** field is set to YES, setting a lower Desktop resolution in the System Setup might fix this.

DIVE might not work on some systems when the Desktop is set to 16 million colors. If your system is set to 16 million colors and DIVE does not work properly, try setting the Desktop to fewer colors in the System Setup.

Replacing DLGEDIT

IBM supports the version of the Dialog Editor (DLGEDIT) in this release of the OS/2 Warp Toolkit.

Presentation Manager Considerations

Making modifications to Microsoft Windows programs to enable dynamic data exchange (DDE) communications with PM programs helps facilitate a gradual migration of applications to PM. Not all data formats are automatically converted when using DDE between Windows programs and PM programs (DDE set public).

The following formats are converted automatically by the OS/2 operating system:

```
PM Application          Windows Application
Data Format             Interpretation

PM bitmap               Windows DIB
PM private              Windows private
Text                    Text (codepage 819)
-------------------------------------------
Windows Application     PM Application
Data Format             Interpretation

Text (codepage 819)     Text
Windows DIB             PM Bitmap
Windows private         PM Private
```

Note: Code page translation (Windows 819 to/from current PM code page) is performed for topic name in all cases.

When data conversion is not automatically performed, programs can still communicate using DDE when the two programs perform the data conversion and pass private data formats.

SOM Considerations

This version of the OS/2 Warp Toolkit contains a subset of the SOMobjects Developer Toolkit. The following SOM programming considerations are briefly described:

- Distributed SOM
- SNGLICLS.HH
- SOM 1.0 OIDL
- SOM Bindings
- SOM Coding Styles
- SOM Compiler
- SOM DLLs

Distributed SOM (DSOM)

New features, known limitations, and restrictions pertain to Distributed SOM (DSOM). They are briefly described as follows:

- DSOM now provides a PM version of the REGIMPL tool for registering servers in the implementation repository. It is called PREGIMPL and is similar in functionality to REGIMPL. To invoke the tool, type PREGIMPL on a command line. Remember to select **File Save** before exiting to save any changes you make.

- You are now able to control the number of request threads created per server. The environment variable SOMDNUMTHREADS is used to indicate the maximum size of the thread pool. If this environment variable is not set, a separate thread will be created for each request.

- The OUT_LIST_MEMORY, IN_COPY_VALUE, and DEPENDENT_LIST flags, used with the dynamic invocation interface, are not supported.

- Concurrent updates to the implementation repository are currently not properly serialized and can conflict.

- The is_nil method of SOMDObject has been changed from a true method to a procedure so that is_nil can be safely invoked on a NULL object pointer. As a result, the syntax for invoking is_nil from C++ client programs has changed. The new syntax is:

  ```
  obj->is_nil(obj,env);
  ```

  Rather than:

```
obj->is_nil(env);
```

where:

obj	Is an object pointer of type SOMDObject.
env	Is of type Environment.

SNGLICLS.HH

The Direct-to-SOM version of the SNGLICLS header file cannot be emitted with the version of the SOM Compiler included with this Toolkit. Therefore, this file (SNGLICLS.HH) is provided as part of the SOM Direct-to-SOM headers.

SOM 1.0 OIDL Users

If you need to recompile an OIDL class that overrides somDumpSelf or somDumpSelfInt, change the data type of the *level* parameter in the function definition in your C source program from INT to LONG. For example, if your original class source program has a somDumpSelfInt override procedure similar to:

```
SOM_Scope void SOMLINK somDumpSelfInt(
     <className> *somSelf, INT level)
{
  ...
}
```

Change it to read:

```
SOM_Scope void SOMLINK somDumpSelfInt(
     <className> *somSelf, LONG level)
{
 ...
}
```

Because both INT and LONG data types require 4 bytes on OS/2, this change does not affect the binary interface of your class.

SOM Bindings

The XH and H files included with this OS/2 Warp Toolkit work only with the IDL files included with this Toolkit. You will need to generate new SOM bindings if you install a new version of SOM. This means that the XH and H files will need to be re-emitted if new versions of the IDL files are made available. There are two steps that you need to take.

If you install version 'Y' of SOM on top of version 'X', you will need to generate SOM bindings for version 'Y'. The version 'X' SOM bindings are not guaranteed to be compatible with version 'Y'.

- Use the SOMSTARS.CMD file to generate the SOMSTARS version of the bindings.
- Use SOMCORBA.CMD to generate the SOMCORBA version of the bindings. This upgrades your SOM bindings.

The command file WPIDL2XH.CMD is provided to upgrade Workplace Shell bindings for developers who will upgrade their SOMobjects Developer Toolkit in the future. This command file emits the .XH header files from the Workplace Shell .IDL files. The file should be invoked upon each of the .IDL files from the OS/2 Warp Toolkit to regenerate the headers for C++. This is only necessary when you upgrade to a new level of the SOMobjects Developer Toolkit. Invoking the SOM compiler's .XH emitter on the Workplace Shell .IDL files do not emit .XH files for you, because the Workplace Shell classes currently only maintain passthru sections for .H files for C.

SOM Coding Styles

There are two possible forms of C bindings for SOM programming:

SOMCORBA	The strict CORBA-compliant form in which pointer references ('*'s) are NOT exposed in object references.
SOMSTARS	The OIDL-compatible C++ form in which pointer references ('*'s) are visible in object references.

The SOMSTARS form is more appropriate if you plan to move your class implementations from C to C++ at some future point. This choice will determine how object references will appear in all of your C programs. For example, to declare a reference to an instance of class Foo, you would code either:

```
Foo afoo;   /* Strict CORBA compliant form          */
or
Foo *afoo; /* C++ migration or OIDL-compatible form */
```

If you later decide to switch from one SOM coding style to the other, you will have to convert any C code that you have already written in one style to the other style.

The Workplace Shell uses the SOMSTARS version of the SOM header files. Therefore, the OS/2 Warp Toolkit installs the SOMSTARS version of the header files.

The Workplace Shell interface definition language (IDL) files (WP*.IDL) are the counterparts for the documented Workplace Shell classes' SC files provided with the OS/2 2.1 Toolkit. The WP*.H and WP*.XH files emitted from the Workplace Shell's IDL files by the SOM compiler are also provided.

The OS/2 makefiles for the Workplace Shell OS/2 Warp Toolkit samples are written assuming the SOMSTARS style of coding. The C samples provided with the SOMobjects Developer Toolkit use the SOMCORBA style of coding (these samples are not part of the IBM Developer's Toolkit for OS/2 Warp).

SOM Compiler

New features, known limitations, and restrictions pertain to the SOM compiler. There are briefly described as follows:

- Mutually recursive IDL *struct* and *union* are not currently supported. The following is an example of unsupported mutual recursion:

```
struct X;
struct Y
{
  sequence<X> indirectSelf;
};
struct X
{
  sequence<Y> indirectSelf;
};
```

- The C bindings do not permit the use of multiple methods with the same name that also take an argument of data type VA_LIST within the same module. For example, the following legal IDL will result in incorrect C usage bindings:

```
module X
{
  interface Y
  {
    void Foo (in LONG f, in VA_LIST ap);
  };
  interface Z
  {
    void Foo (in LONG f, in VA_LIST ap);
```

```
  };
};
```

- The SOM C++ language bindings are built assuming use of the VisualAge C++ for OS/2 compiler, but other C++ compilers should be able to use these bindings as well. For example, to use BCOS2 (the Borland C++ compiler for OS/2), use -DSOMLINK=_syscall on the compile line, and make sure that the SOMobjects' include directory is consulted before BCOS2/include (because BCOS2/include contains older SOM.H include files).
- If the SOM compiler is interrupted by the user (using Ctrl+C, for example), it sometimes leaves a temporary file with a .CTN extension in the temporary directory specified by the SMTMP environmental variable. These temporary files should be removed periodically.
- When direct references to SOMFOREIGN types are made in an IDL *struct* or *union* , the C or C++ language bindings are generated incorrectly. To refer to a SOMFOREIGN type (for example, "somId") in a *struct* or *union* , it is necessary to supply a secondary typedef for "somId." For example:

```
#include <somobj.idl>
struct S1
{
  somId badId;    /* Generates incorrect */
};                /* C/C++ bindings      */
#include <somobj.idl>
typedef somId somId2;
struct S1
{
  somId2 badId;   /* OK                  */
};
```

SOM Dynamic Link Libraries (DLLs)

When SOMobjects 2.0 (*not* the version included with this OS/2 Warp Toolkit) is installed on OS/2 Warp Version 3, the LIBPATH, PATH, and DPATH environment variables in CONFIG.SYS are changed to place the SOMobjects directories before the operating system directories in the search order for DLLs, EXEs, and data files. This causes the Workplace Shell to encounter an unrecoverable error the next time the system is restarted because it requires the SOM DLLs that are contained in the \OS2\DLL subdirectory.

In order to allow SOMobjects 2.0 Workstation applications to run successfully, you must edit CONFIG.SYS and change the LIBPATH, PATH, and DPATH statements before restarting the system. These statements must be changed to place the operating system directories before the SOMobjects directories.

Note: This only applies to SOMobjects Workstation applications, not Workgroup applications.

The same problem will occur for any application that includes SOMobjects 2.0 DLLs if the application places its DLL directory first in the LIBPATH. Again, the workaround is to assure that the \OS2\DLL subdirectory is before any other directory that contains earlier versions of the SOM DLLs. Any changes made to the PATH and DPATH environment variables by the application installation must also be reversed.

Workplace Shell Considerations

When including an OBJECTID=<....> keyname=value pair in a setup string, you must specify it at the end of the setup string.

Toolkit Support

The OS/2 Developer's Toolkit provides programming tools, samples and information to support end-user development of OS/2 applications. We will make every effort to respond to support requests, but the Toolkit is provided on an as-is basis, and there is no guarantee of formal

support.

Informal electronic support for the IBM Developer's Toolkit is provided through the Internet and the OS/2 BBS. Internet users may report problems by sending mail to devcon@us.ibm.com. Users may obtain informal support or exchange ideas or comments by accessing the DEVCON CFORUM on the OS/2 BBS under Talklink (a feature under the IBMLink Commercial Services). For TalkLink access, U.S. customers can call 1-800-547-1283; other customers can contact their IBM Marketing Representatives.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS information "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AVC	Presentation Manager
C/2	SOMobjects
Common User Access	Ultimotion
C Set ++	VisualAge
CUA	WIN-OS2
IBM	VoiceType
Workplace Shell	OS/2

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Toolkit Licensing Information

The IBM OS/2 Developer's Toolkit includes header files, include files, and audio files. These files have extensions of .H, .HH, .XH, .INC, and .WAV. The Toolkit also contain programming tutorials illustrating OS/2 and multimedia programming techniques. Both executable and source files for these tutorials are included. You may copy and distribute the above header files, include files, audio files, and programming turorials in any form without payment to IBM, for the sole purposes of developing, using, marketing, and distributing application programs written to the OS/2 and MMPM/2 application programming interface. Each copy or any portion of these programming tutorials or any derivative work thereof, which are distributed to others, must include a copyright notice as follows: "(C) Copyright (your company name) (year). All Rights Reserved." The IBM Developer's Toolkit for OS/2 Warp includes a dynamic link library, SMAPI.DLL, for the Voice Type application programming interfaces. You may copy and distribute this library in object code form only, as part of your OS/2 software application program, without payment to IBM, provided that (1) you agree to indemnify, hold harmless and defend IBM against any claims, lawsuits and liabilities arising out of the use and distribution of your applications and (2) you distribute your applications pursuant to a license that prohibits the user from copying (except for backup purposes) or distributing this library.
